Algoritmos e Estrutura de Dados II (AE23CP-3CP)

Aula #03 - Complexidade de Algoritmos - Parte II

Profª Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

Complexidade de Algoritmos - Parte II

Conceitos

Análise de Algoritmos

Área da computação que visa determinar a complexidade (custo) de um algoritmo, o que torna possível:

- Comparar algoritmos
- Determinar se um algoritmo é "ótimo".

Custo de um algoritmo:

- Tempo (número de passos)
- Espaço (memória)

Critérios de Complexidade

Como é medida?

A complexidade de um algoritmo é medida segundo um **modelo matemático** que supõe que este vai trabalhar sobre uma entrada (massa de dados) de tamanho N.

Execução de um algoritmo

É dividido em etapas elementares - **passos** *(número fixo de operações básicas, tempo constante, operação de maior freqüência chamada* **dominante***)* .

- O número de passos de um algoritmo é considerado como o número de execuções da operação dominante em função das entradas, desprezando-se constantes aditivas ou multiplicativas.

Função de avaliação do tempo de execução

Expressão matemática que fornece o **número de passos** efetuados pelo algoritmo a partir de uma certa **entrada.**

Critérios de Complexidade - exemplos

Ex.: Soma de vetores

```
for(i=0; i<n; i++){
  s[i] = x[i] + y[i];
}
```

Número de passos = número de somas (n)

Ex.: Soma de matrizes

```
for(i=0; i<n; i++){
  for(j=0; j<n; j++){
          c[i][j] = a[i][j] + b[i][j]
        }
}
```

Número de passos = número de somas (n*n)

Critérios de Complexidade - exemplos

Ex.: Produto de matrizes

```
for(i=0; i<n; i++){
 for(j=0; j<n; j++){
  p[i][j] = 0;
  for(k=0; k<n; k++){
   p[i][j] = p[i][j] + a[i][k*]b[k][j];
  }
 }
}
```

Número de passos = número de somas (n*n*n)

Critérios de Complexidade

Operação fundamental

operação escolhida para medir a quantidade de trabalho realizado por um algoritmo

- Às vezes, é necessária mais de uma operação fundamental e com pesos diferentes: **custo**

Tamanho da entrada:

É associado as estruturas de dados do algoritmo;

- Quantidade de bits para representar um número;
- Quantidade de nós e arestas em um grafo;
- Tamanho do vetor em um problema de ordenação

Análise no melhor caso, pior caso e caso médio

- Exemplo: Algoritmo Busca Sequencial

Tamanho da entrada

```
Algoritmo Busca_Seq(int vet[1..n], chave )
1. i ← 0 ; achou ← F ;                          { inicialização }
2. repita                                       {iteração}
3      i ← i + 1 ;                              {incrementa índice}
4.     se vet[i] = chave então achou ← V; fim-se {compara com chave}
5. até (achou ∨ i = n) ;                        {fim iteração}
6. se achou então pos ← i fim-se ;              {verifica se achou}
7. retorne-saída( achou, pos )                  {retorna saída}
8. fim-algoritmos                               {fim do algoritmo}
```

Operação fundamental

Análise no melhor caso

Considere $chave = 10$

i	1	2	3	4	5	6	7	8	9	10
vet	10	5	12	9	3	2	14	98	1	4

para $i = 1$, $vet[i] = chave (vet[1] = chave = 10)$

Conclusão:

1. O algoritmo precisa de uma execução da operação fundamental para resolver o problema
2. pouco interessa prático, muito otimista!

Análise no pior caso

Considere $chave = 4$

i	1	2	3	4	5	6	7	8	9	10
vet	10	5	12	9	3	2	14	98	1	4

para $i = 10$, $vet[i] = chave(vet[10] = chave = 4)$

Conclusão:

1. O algoritmo encontra o elemento na última posição pesquisa. Realizou o **máximo** de esforço computacional.
2. considerado também **complexidade pessimista**, porém de grande interesse prático!

Análise no caso médio

Considera a probabilidade de todas as entradas

i	1	2	3	4	5	6	7	8	9	10
vet	10	5	12	9	3	2	14	98	1	4

- $i = 1$, basta uma execução da operação fundamental
- $i = 2$, duas execuções da operação fundamental
- $i = 3$, três execuções da operação fundamental
- ...
- $i = 10$, dez execuções da operação fundamental

Portanto:

$(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10)/10 \approx 5$

Para um vetor de n posições temos: $n/2$

Conclusão:

- também denominada complexidade esperada.
- É a mais indicada para muitos algoritmos do dia-a-dia, porém, em muitos casos, seu cálculo é extremamente complexo

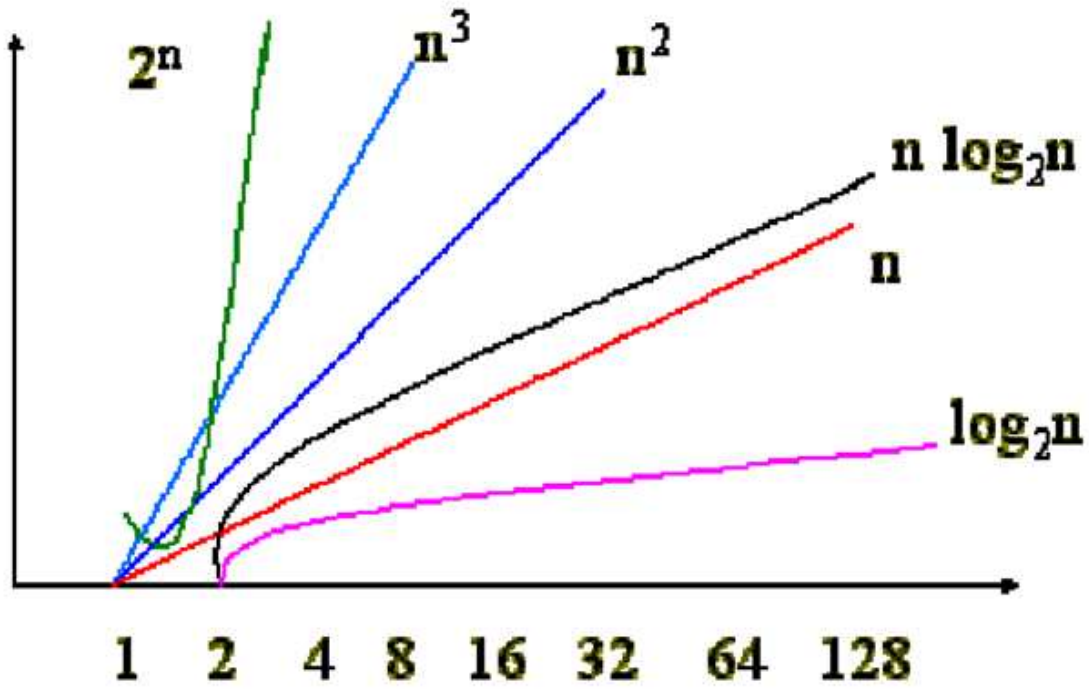
Ordens Assintóticas

Complexidade assintótica

Definida pelo crescimento da complexidade para **entradas suficientemente grandes**.

- É expressa por uma função. Exemplo:
  - n, $log\ n$, n^2, 2^n
- Obs.: Em complexidade de algoritmos convenciona-se representar uma função constante k por 1

Ordens Assintóticas



Relação entre ordens

$O(1) < O(log\ n) < O(n) < O(n\ log\ n) < O(n^2) < O(2^n)$

Ordens Assintóticas - Cota Superior

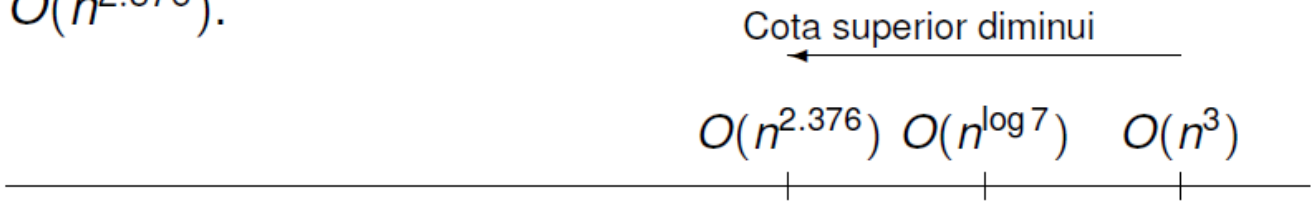
- Seja dado um problema, por exemplo, multiplicação de duas matrizes quadradas $n \times n$.
- Conhecemos um algoritmo para resolver este problema (pelo método trivial) de complexidade $O(n^3)$.
- Sabemos assim que a complexidade deste problema não deve superar $O(n^3)$, uma vez que existe um algoritmo que o resolve com esta complexidade.
- Uma cota superior ou limite superior (*upper bound*) deste problema é $O(n^3)$.

- A cota superior de um problema pode mudar se alguém descobrir um outro algoritmo melhor.

Ordens Assintóticas - Cota Superior

- O Algoritmo de Strassen reduziu a complexidade para $O(n^{\log 7})$. Assim a cota superior do problema de multiplicação de matrizes passou a ser $O(n^{\log 7})$.

- Coppersmith e Winograd melhoraram ainda para $O(n^{2.376})$.



- Note que a cota superior de um problema depende do algoritmo. Pode diminuir quando aparece um algoritmo melhor.

Analogia com *Record Mundial*

A cota superior para resolver um problema é análoga ao *record mundial* de uma modalidade de atletismo. Ele é estabelecido pelo melhor atleta (algoritmo) do momento. Assim como o *record mundial*, a cota superior pode ser melhorada por um algoritmo (atleta) mais veloz.

"Cota superior" da corrida de 100 metros rasos:

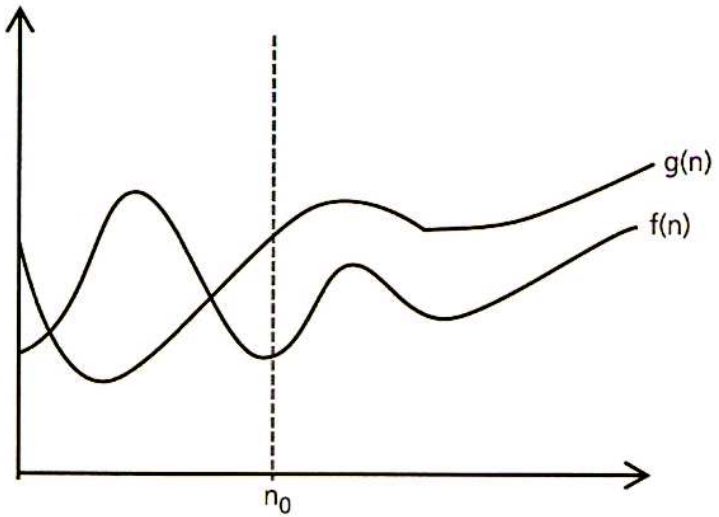
Ano	Atleta (Algoritmo)	Tempo
1988	Carl Lewis	9s92
1993	Linford Christie	9s87
1993	Carl Lewis	9s86
1994	Leroy Burrell	9s84
1996	Donovan Bailey	9s84
1999	Maurice Greene	9s79
2002	Tim Montgomery	9s78
2007	Asafa Powell	9s74
2008	Usain Bolt	9s72
2008	Usain Bolt	9s69
2009	Usain Bolt	9s58

Cota Assintótica Superior (CAS)

Uma CAS é uma função que cresce mais rapidamente do que outra: está acima, a partir de certo ponto.

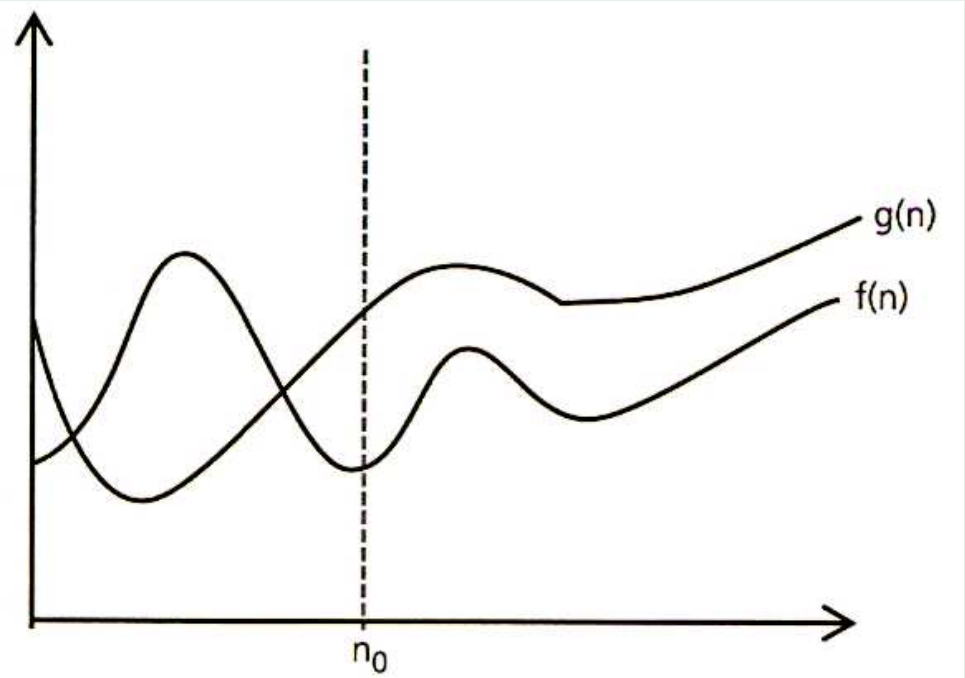
- Define-se g uma cota assintótica superior para f se e somente se: $(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(f(n) \leq g(n))$,

  significando que, para um n suficientemente grande, $g(n)$ domina $f(n)$.



$g(n)$ é CAS para $f(n)$

Cota Assintótica Superior (CAS)



Considerando as funções $f(n) = n + 4$ e $g(n) = n^2$;

- para: $n = 0, 1, 2$, temos $f(n) > g(n)$;
- mas para $n = 3$, temos $f(n) < g(n)$.

- **Cota Assintótica Superior (CAS).** Provar que $g(n) = n^2$ é CAS para $f(n) = n + 4$;

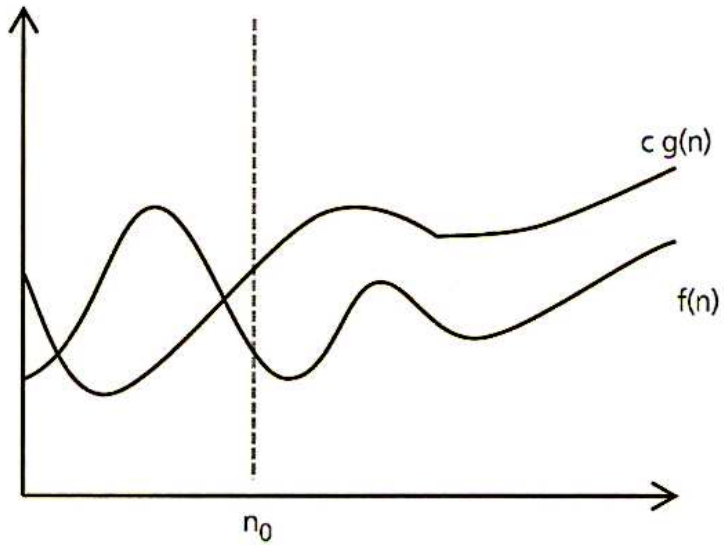
n	**n + 4**	**n^2**
0	$0 + 4 = 4$	$0 \times 0 = 0$
1	$1 + 4 = 5$	$1 \times 1 = 1$
2	$2 + 4 = 6$	$2 \times 2 = 4$
3	$3 + 4 = 7$	$3 \times 3 = 9$
4	$4 + 4 = 8$	$4 \times 4 = 16$
5	$5 + 4 = 9$	$5 \times 5 = 25$

Ordens Assintóticas

Notação O ("big oh"): define uma CAS:

$f(n)$ é $O(g(n))$ se e somente se para alguma constante $c \in \mathbb{R}_+) : c.g(n)$ é CAS para $f(n)$, isto é,

$$(\exists c \in \mathbb{R}_+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(f(n) \leq c.g(n))$$



Exemplo:

$f(n) = 5n$ e $g(n) = n^2$

Exemplo:

- Sejam $f(n) = 5n$ e $g(n) = n^2$

  Seja $c = 1$ e $n_0 = 5$

n	**5n**	**n^2**
0	$5 \times 0 = 0$	$0 \times 0 = 0$
1	$5 \times 1 = 5$	$1 \times 1 = 1$
2	$5 \times 2 = 10$	$2 \times 2 = 4$
3	$5 \times 3 = 15$	$3 \times 3 = 9$
4	$5 \times 4 = 20$	$4 \times 4 = 16$
5	$5 \times 5 = 25$	$5 \times 5 = 25$
6	$5 \times 6 = 30$	$6 \times 6 = 36$

Ordens Assintóticas

Notação O

permite dizer que uma função de n é "**menor ou igual a**" outra função pela desigualdade na definição, descontando-se um fator constante (a constante c da definição).

Exemplos

- Método Bolha no Pior Caso: $O(n^2)$
- Método Mergesort no Pior Caso: $O(n\ log\ n)$
- Pesquisa Sequencial no Pior Caso: $O(n)$
- Pesquisa Binária no Pior Caso: $O(log\ n)$

Ordens Assintóticas - Nomes de Classe O

classe	nome
$O(1)$	constante
$O(\lg n)$	logarítmica
$O(n)$	linear
$O(n \lg n)$	$n \log n$
$O(n^2)$	quadrática
$O(n^3)$	cúbica
$O(n^k)$ com $k >= 1$	polinomial
$O(2^n)$	exponencial
$O(a^n)$ com $a > 1$	exponencial

Ordens Assintóticas - Cota Inferior

- As vezes é possível demonstrar que, para um dado problema, **qualquer** que seja o algoritmo a ser usado, o problema requer pelo menos um certo número de operações.
- Essa complexidade é chamada **cota inferior (lower bound)** do problema.
- Note que a cota inferior depende do problema mas não do particular algoritmo.

Exemplo: multiplicação de matrizes

- Para o problema de multiplicação de matrizes quadradas $n \times n$, apenas para ler os elementos das duas matrizes de entrada ou para produzir os elementos da matriz produto leva tempo $O(n^2)$.
  - Assim uma cota inferior trivial é $\Omega(n^2)$.

Na analogia anterior, uma cota inferior de uma modalidade de atletismo não dependeria mais do atleta. Seria algum tempo mínimo que a modalidade exige, qualquer que seja o atleta.

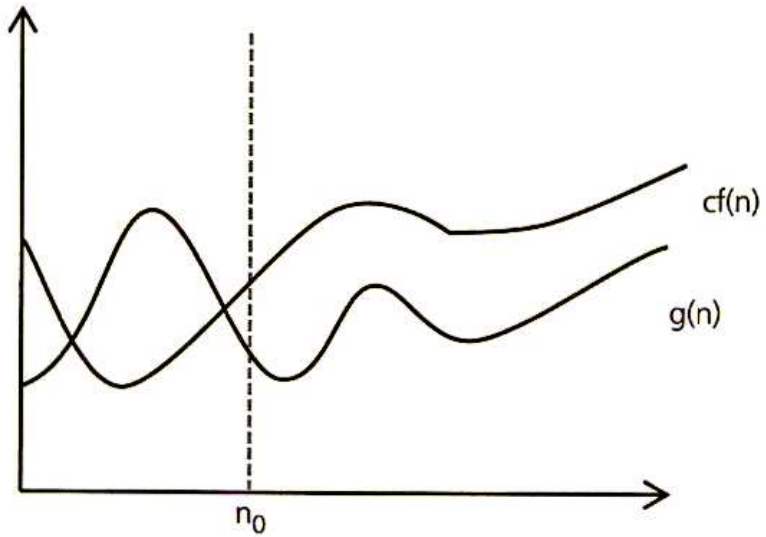
- **Exemplo:** uma cota inferior trivial para os 100 metros rasos seria o tempo que a velocidade da luz leva para percorrer 100 metros no vácuo.

Ordens Assintóticas

Notação Ω ("big Ômega"): define uma **cota assintótica inferior**

$f(n)$ é $\Omega(g(n))$ se e somente se para alguma constante $c \in \mathbb{R}_+$) : $c.f(n)$ é CAS para $g(n)$, isto é,

$$(\exists c \in \mathbb{R}_+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \leq c.f(n))$$



Exemplo:

Considere $f(n) = n^3$ e $g(n) = 6n^2 + 10$, $g(n)$ é $\Omega(f(n))$

Notação Ω ("big Ômega") - Considerações

- Os algoritmos de $O(1)$, $O(n)$, $O(n^2)$, . . . são polinomiais.
- Já 2^n é uma função não limitada por polinômio.
- Assim, um algoritmo de $\Omega(2^n)$, $\Omega(3^n)$, . . . é dito **exponencial**.

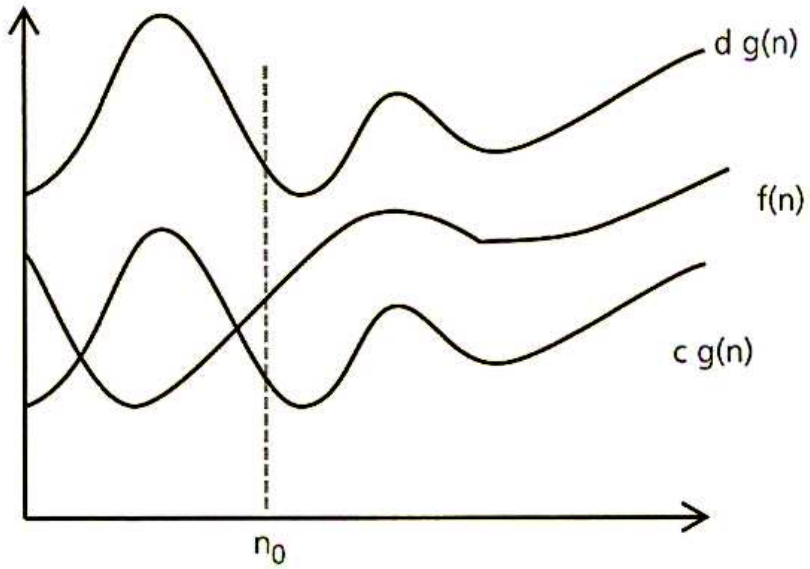
Logo, se o melhor algoritmo conhecido tem limite Ω, o problema é considerado **intratável**.

Ordens Assintóticas

Notação Θ ("big Theta"): define um **limite assintótico exato**

$f(n)$ é $\Theta(g(n))$ se e somente se para alguma constante $c \in \mathbb{R}_+$) : $c.f(n)$ é CAS para $g(n)$, ie,

$$(\exists c, d \in \mathbb{R}_+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(c.g(n) \leq f(n) \wedge f(n) \leq d.g(n))$$



Exemplo:

Considere $f(n) = 7n + 3$ e $g(n) = 2n + 5$

Notação Θ ("big Theta")
Exemplo: busca por maior elemento

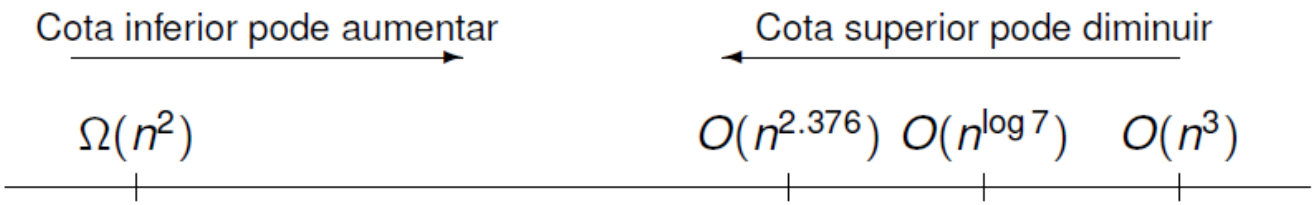
```
Algoritmo Max(int vet[1..n])
        maior <- vet[1];
        para i de 2 até n faça
            se vet[i] > maior então
                    maior <- vet[i]
fim_algoritmo
```

Exemplo:

Considere que os algoritmos A é $\Theta(n)$, A' é $\Theta(n^2)$ e A'' é $\Theta(2^n)$. E um passo de execução de 0,0001 segundo.

Tempo Total de Execução				
		Tamanho dos Dados de Entrada n		
Algoritmo	Ordem	10	50	100
A	n	0,001 seg	0,005 seg	0,01 seg
A'	n^2	0,01 seg	0,25 seg	1 seg
A''	2^n	0,1024 seg	3.570 anos	4×10^{16} séculos

Ordens Assintóticas - Meta: aproximando as duas cotas



- Se um algoritmo tem uma complexidade que é igual à cota inferior do problema, então ele é assintoticamente ótimo.
- O algoritmo de Coppersmith e Winograd é de $O(n^{2.376})$ mas a cota inferior (conhecida até hoje) é de $\Omega(n^2)$. Portanto não podemos dizer que ele é ótimo.
- Pode ser que esta cota superior possa ainda ser melhorada. Pode também ser que a cota inferior de $\Omega(n^2)$ possa ser melhorada (isto é "aumentada"). Para muitos problemas interessantes, pesquisadores dedicam seu tempo tentando encurtar o intervalo ("gap") até encostar as duas cotas.

Princípio da Absorção

- Para funções f e g de $\mathbb{N}$ em $\mathbb{R}_+$, se f é absorvida por g, sua soma pontual: $f + g$ é $\Theta(g)$.
  - Assim, como $O(n\ log\ n)$ é absorvida por $O(n^2)$, a soma $O(nlogn + n^2)$ é $O(n^2)$.

Princípio da Absorção

Exemplo:

Considere a função quadrática $3n^2 + 10n + 50$:

n	$3n^2 + 10n + 50$	$3n^2$
64	12978	12288
128	50482	49152
512	791602	786432
1024	3156018	3145728
2048	12603442	12582912
4096	50372658	50331648
8192	201408562	201326592
16384	805470258	805306368
32768	3221553202	3221225472

- Como se vê, $3n^2$ é o termo dominante quando n é grande.
- De um modo geral, podemos nos concentrar nos termos dominantes e esquecer os demais.

Cálculo da complexidade de algoritmos

Cálculo da complexidade de algoritmos

Como calcular a complexidade das seguintes estruturas algorítmicas?

- Atribuição: **v** $\leftarrow$ **e**,
- Condicional: **se** b **então** S **senão** T (ou **se** b **então** S),
- Seqüência (ou composição): **S** ; **T**,
- Iteração definida (ou incondicional): **para** i **de** j **até** m **faça S**,
- Iteração indefinida (ou condicional): **enqto** b **faça** S.

Cálculo da complexidade de algoritmos

Atribuição

- A atribuição **v** $\leftarrow$ **e** pode ser:
  - simples, como a atribuição de um valor a uma variável,
  - complexa, como a atualização de uma matriz, uma inserção de um nodo num grafo, uma inserção de um elemento num conjunto

Além disso, ela costuma requerer uma avaliação de **e**, que pode ser uma expressão ou mesmo uma função.

Atribuição - exemplo:

Considere as atribuições a seguir:

a) Para variáveis inteiras i e j:

i $\leftarrow$ 0; = $O(1)$

j $\leftarrow$ i ; = $O(1)$

b) Para lista v de inteiros e variável inteira m:

m $\leftarrow$ Max(v) valor máximo.

- Esta atribuição envolve (sendo n o comprimento da lista em v):
  1. determinar o máximo da lista v, com complexidade $O(n)$;
  2. transferir este valor, com complexidade $O(1)$.

Atribuição - exemplo:

Considere as atribuições a seguir:

c) Para listas u, v e w:

u $\leftarrow$ v transfere lista

w $\leftarrow$ Reversa(v) inverte lista.

- A atribuição de transferência de cada elemento da lista v, para uma lista u com comprimento n, tem complexidade $O(n)$.
- A atribuição w $\leftarrow$ Reversa(v) envolve:
  - inverter a lista v, com complexidade $O(n)$;
  - transferir os elemento da lista invertida, com complexidade $O(n)$.

Cálculo da complexidade de algoritmos

Complexidades constantes: $O(1)$

- Leituras, escritas, funções matemáticas.

Exemplos:

leia(x); $= O(1)$

escreva("Digite uma tecla"); $= O(1)$

x = a mod b; $= O(1)$

y = ((a+b)/(y*z)+(var1 * var2)); $= O(1)$

- **Exceção:**

escreva(Max(n)); $= O(n)$

Cálculo da complexidade de algoritmos

Condicionais

- As formas mais usuais da estrutura condicional são as seguintes:

se b **então** S **senão** T **fim-se**

e

se b **então** S **fim-se**.

Condicionais - exemplo:

Considere a atribuição a seguir:

a) Para variável inteira i:

se $(i = 0)$ **então**
$i \leftarrow i + 1$.

- Esta estrutura condicional envolve:
  - determinar se o valor de i é 0, com complexidade $O(1)$;
  - se sim, executar a atribuição $i \leftarrow i + 1$, com complexidade $O(1)$.

Condicionais - exemplo:

Considere as atribuição a seguir:

b) Para lista v de inteiros e variável inteira m:

```
se (m = 0) então
    m ← Max(v).
```

- Esta atribuição envolve:
  - determinar se o valor de m é 0, com complexidade $O(1)$;
  - se sim, executar a atribuição m ← Max(v), com complexidade $O(n)$.

c)

```
se Max(v) = 0 então
   v ← Ordene(v)
fim-se.
```

$= O(n + n^2) = O(n^2)$

Cálculo da complexidade de algoritmos

Sequência (ou composição)

- O desempenho da seqüência **S ; T** é a soma dos desempenhos de suas componentes.

Exemplos:

a) Para variáveis inteiras i e j:
   i $\leftarrow$ 0 ; j $\leftarrow$ i.
   Sua complexidade tem ordem 1 + 1: $O(1)$.

b) Para lista v de inteiros e variável inteira m:
   m $\leftarrow$ Max(v) ; m $\leftarrow$ m + 1.
   Sua complexidade tem ordem $n + 1$: $O(n)$.

c) Para listas u, v e w:
   u $\leftarrow$ v ; w $\leftarrow$ Reversa(v).
   Sua complexidade tem ordem $n + n$: $O(n)$.

Cálculo da complexidade de algoritmos

Iteração definida

Com preservação assintótica de tamanho:

para i **de** j **até** m **faça** S

Exemplos:

- Exemplo:

a) para i de 1 até 20 faça m $\leftarrow$ m + 1 (sem preservação assintótica)
$= O((20 - 1 + 1).O(1) = O(20) \equiv O(1)$

b) para i de 1 até n faça A[i] $\leftarrow$ 0
$= O((n - 1 + 1).O(1)) = O(n)$

c) para i de 1 até n faça A[i] $\leftarrow$ A[i] + 1
$= O((n - 1 + 1).O(1)) = O(n)$

Cálculo da complexidade de algoritmos

Iteração definida - exemplo:

- Qual a complexidade do algoritmo classificação de vetor?

Algoritmo 3.3.3: Classificação de vetor.

```
Procedimento Classif_Max( A : V )          {classifica vetor em ordem ascendente}
    {Entrada: vetor A : V.
     Saída: vetor A : V (o vetor de entrada classificado em ordem ascendente).}
  sorte V := vetor [ 1..n ] de Elmord ;
  var {auxiliar} Temp : Elmord {temporária (para troca)} ;
1. para i de 1 até n    faça                                   {de 1 a 10}
2.   para j de 1 até n - i faça                                {de 2 a 9}
3.     se A[ j ] > A[ j + 1 ]                                  {testa inversão}
4.       então                                                 {troca A[ j ] com A[ j + 1 ]}
5.         Temp ← A[ j ] ;
6.         A[ j ] ← A[ j + 1 ] ;
7.         A[ j + 1 ] ← Temp ;
8.     fim-se ;                                                {3: A[ j ] ← A[ j + 1 ]?}
9.   fim-para ;                                                {2: j de 1 até n - i}
10. fim-para ;                                                 {1: i de 1 até n}
11. retorne-saída( A ) ;                                       {dá saída A}
12. fim-Procedimento   .                                       {fim de Classif_Max: classificação ascendente}
```

Cálculo da complexidade de algoritmos

Iteração Indefinida

- **enquanto** b **faça** S

Exemplos:

a) Para variável inteira i:

i $\leftarrow$ 0;

enqto i $<$ 10 faça i $\leftarrow$ i $+$ 1;

✓ Sua complexidade tem ordem: $O(1)$.

b) Inicialização de vetor A[1..n] de inteiros:

i $\leftarrow$ 0;

enqto i $<$ n faça i $\leftarrow$ i $+$ 1 ; A[i] $\leftarrow$ 0

fim-enqto

✓ Sua complexidade tem ordem: $O(n)$.

Cálculo da complexidade de algoritmos

Iteração Indefinida

```
escreva("digite um valor:")
    ;
leia(valor);
enquanto (valor != -1) faça
        leia(valor);
fim_enquanto.
```

Complexidade = ???

Iteração indefinida - exemplo *Insertion Sort*

- Qual a complexidade do algoritmo *Insertion Sort* ???

INSERTION-SORT(A, n)	Tempo
1 **para** $j \leftarrow 2$ **até** n **faça**	$\Theta(n)$
2 $\quad chave \leftarrow A[j]$	$\Theta(n)$
3 $\quad \triangleright$ Insere $A[j]$ em $A[1 \ldots j-1]$	
4 $\quad i \leftarrow j-1$	$\Theta(n)$
5 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**	$nO(n) = O(n^2)$
6 $\quad\quad A[i+1] \leftarrow A[i]$	$nO(n) = O(n^2)$
7 $\quad\quad i \leftarrow i-1$	$nO(n) = O(n^2)$
8 $\quad A[i+1] \leftarrow chave$	$O(n)$

Consumo de tempo: $O(n^2)$

Cálculo da complexidade de algoritmos

Iteração indefinida - exemplo:

- Qual a complexidade do Algoritmo da Loteria esportiva?

Algoritmo 3.3.4: Loteria esportiva.

```
Procedimento Loto( Rslt : V, Apst : M )                    {teste da loteria esportiva}
  {Entrada: vetor Rslt : V (resultado de teste), matriz Apst : M (apostador)
     Saída: vetor Ptos : vetor [ 1..n ] de IN (número de pontos feitos por apostador).}
   sorte V := vetor [ 1..13 ] de { 1..3 } ; M := matriz [ 1..n, 0..13 ] de Info ;
1. i ← 1 ;                                          {inicializa número de apostadores}
2. enqto i ≤ n faça                                 {examina apostador: de 2 a 8}
3.    Ptos[ i ] ← 0 ;                               {inicializa número de pontos do apostador i}
4.    para j de 1 até 13 faça                       {examina palpites: de 4 a 6}
5.       se Apst[ i,j ] = Rslt[ j ] então Ptos[ i ] ← Ptos[ i ] + 1     .
6.    fim-para ;                                    {4: pontos do apostador i contabilizados}
7.    i ← i + 1 ;                                   {novo apostador}
8. fim-enqto ;                                      {2: contabilizados os pontos de todos apostadores}
9. para i de 1 até n faça                           {lista ganhadores: de 9 a 11}
10.  se Ptos[ i ] = 13 então escreva( Apst[ i,0 ], "Ganhador, parabéns" )
11. fim-para ;                                      {9: ganhadores listados}
12. retorne-saída( Ptos ) ;                         {dá saída Ptos}
13. fim-Procedimento .                              {fim do algoritmo Loto: loteria esportiva}
```

Referências

- *Complexidade de Algoritmos*. Laira Toscani e Paulo Veloso. Ed. Sagra-Luzzatto. 2001.
- *Projeto de Algoritmos com Implementação em Pascal e C*. Nívio Ziviani. Ed. Thomson. 2004.
- *Fundamentos Matemáticos para a Ciência da Computação*. Ed. LTC. 2008.