Análise de Algoritmos

Prof.: Maiquel de Brito
maiquel.b@ufsc.br

BLU3202 - Algoritmos e Estruturas de Dados
Engenharia de Controle e Automação
Departamento de Engenharias
UFSC Blumenau

Table of contents

Introdução

Análise de Algoritmos

Diferentes algoritmos podem resolver o mesmo problema

ex.: algoritmos de ordenação, pesquisa etc

Como medir se um algoritmo é eficiente?

Tempo de execução, consumo de memória...

dependem da máquina que os executa

Solução: analisar a quantidade de instruções executadas em função do **tamanho da entrada**

Tamanho da entrada

Tamanho de entrada fixo:

```
1 void dobro(int n){
2     return n*2;
3 }
```

Tamanho de entrada variável:

```
1 void dobro(int *v, int n){
2     for(int i=0;i<n;i++){
3         printf("%d",v[i]*2);
4     }
5 }
```

Desempenho em função do tamanho da entrada (n)

```
1  void teste(int *v, int n){                          custo   vezes
2      for(int i=0;i<n;i++){. . . . . . . . . . . . . . c1      n
3          printf("%d - ",v[i]); . . . . . . . . . . . c2      n
4      }
5  }
6
7  void main(){
8      int vetor[] = {1,2,3,4,5};
9      int tamanho = sizeof(vetor)/sizeof(int);
10     teste(vetor,tamanho);
11 }
```

Complexidade da função `teste`

$$
\begin{aligned}
C(n) &= c_1 \times n + c_2 \times n \\
&= c_1 n + c_2 n \\
&= (c_1 + c_2) n
\end{aligned}
$$

Desempenho em função do tamanho da entrada (n)

```
1  void teste(int *v, int n)                          custo    vezes
2      for(int i=0;i<n;i++){                           c1         n
3          for(int j=0;j<n;j++){                       c2       n x n
4              printf("%d - ",v[i]+j);                 c3       n x n
5          }
6      }
7  }
8
9  void main(){
10     int vetor[] = {1,2,3,4,5};
11     int tamanho = sizeof(vetor)/sizeof(int);
12     teste(vetor,tamanho);
13 }
```

$$
\begin{aligned}
C(n) &= c_1 \times n + c_2 \times n \times n + c_3 \times n \times n \\
&= c_1 n + c_2 n^2 + c_3 n^2 \\
&= (c_2 + c_3) n^2 + c_1 n
\end{aligned}
$$

Desempenho em função do tamanho da entrada (n)

Qual é o desempenho da função `primeiro_par`?[1]

```
1 int primeiro_par(int *v, int n){
2     for(int i=0;i<n;i++){
3         if(v[i]%2==0){
4             return i;
5         }
6     }
7     return -1;
8 }
```

[1] A cláusula `return` (linhas 4 e 8) provoca o encerramento da execução do algoritmo.

Desempenho em função do tamanho da entrada (n)

Qual é o desempenho da função `primeiro_par`?

```
1 int primeiro_par(int *v, int n){                    custo  vezes
2     for(int i=0;i<n;i++){                             c1     x
3         if(v[i]%2==0){                                c2     x
4             return i;                                 c3     1
5         }
6     }
7     return -1;                                        c4     1
8 }
```

Desempenho em função do tamanho da entrada (n)

Qual é o desempenho da função `primeiro_par`?

```
1  int primeiro_par(int *v, int n){                    custo  vezes
2      for(int i=0;i<n;i++){                            c1     x
3          if(v[i]%2==0){                               c2     x
4              return i;                                c3     1
5          }
6      }
7      return -1;                                       c4     1
8  }
```

x – depende da posição do primeiro número par dentro do vetor

melhor caso: menor tempo de execução sobre todas as possíveis entradas de tamanho n

Desempenho em função do tamanho da entrada (n)

Qual é o desempenho da função `primeiro_par`?

```
1 int primeiro_par(int *v, int n){                    custo  vezes
2     for(int i=0;i<n;i++){                             c1     x
3         if(v[i]%2==0){                                c2     x
4             return i;                                 c3     1
5         }
6     }
7     return -1;                                        c4     1
8 }
```

x – depende da posição do primeiro número par dentro do vetor

melhor caso: menor tempo de execução sobre todas as possíveis entradas de tamanho n

Na função `primeiro_par`, o melhor caso é quando há um número par na primeira posição do vetor. Ex.: `v ={2,3,4,5,6,7,8}`

Nesse caso, $x = 1$ e o custo $C(n) = c_1 \times 1 + c_2 \times 1 + c_3 \times 1 = 3$

Desempenho em função do tamanho da entrada (n)

Qual é o desempenho da função `primeiro_par`?

```
1  int primeiro_par(int *v, int n){                    custo  vezes
2      for(int i=0;i<n;i++){                            c1     x
3          if(v[i]%2==0){                               c2     x
4              return i;                                c3     1
5          }
6      }
7      return -1;                                       c4     1
8  }
```

$\rightsquigarrow$ Assume-se que cada instrução i têm custo $c_i = 1$

melhor caso:[2]

$$C(n) = c_1 \times 1 + c_2 \times 1 + c_3 \times 1$$
$$= 1 \times 1 + 1 \times 1 + 1 \times 1 = 3$$

[2] No melhor caso, a instrução c_4 nunca é executada e, por isso, não é incluída no cálculo

Desempenho em função do tamanho da entrada (n)

Qual é o desempenho da função `primeiro_par`?

```
1  int primeiro_par(int *v, int*n){                    custo  vezes
2      for(int i=0;i<n;i++){                            c1     x
3          if(v[i]%2==0){                               c2     x
4              return i;                                c3     1
5          }
6      }
7      return -1;                                       c4     1
8  }
```

Desempenho em função do tamanho da entrada (n)

Qual é o desempenho da função `primeiro_par`?

```
1  int primeiro_par(int *v, int*n){                    custo  vezes
2      for(int i=0;i<n;i++){                            c1     x
3          if(v[i]%2==0){                               c2     x
4              return i;                                c3     1
5          }
6      }
7      return -1;                                       c4     1
8  }
```

x – depende da posição do primeiro número par dentro do vetor

pior caso: maior tempo de execução sobre todas as possíveis entradas de tamanho n

Desempenho em função do tamanho da entrada (n)

Qual é o desempenho da função `primeiro_par`?

```
1 int primeiro_par(int *v, int*n){                    custo  vezes
2     for(int i=0;i<n;i++){                           c1     x
3         if(v[i]%2==0){                              c2     x
4             return i;                               c3     1
5         }
6     }
7     return -1;                                      c4     1
8 }
```

x – depende da posição do primeiro número par dentro do vetor

pior caso: maior tempo de execução sobre todas as possíveis entradas de tamanho n

Na função `primeiro_par`, é (i) quando o primeiro número par está na última posição ou (ii) quando não há número par no vetor. Ex.: `v={3,5,7,9,11}`

Nesse caso, $x = n$ e o custo $C(n) = 2n + 1$

$$\begin{aligned} C(n) &= c_1 \times n + c_2 \times n + c_4 \times 1 \\ &= 1 \times n + 1 \times n + 1 \times 1 \\ &= 2n + 1 \end{aligned}$$

Desempenho em função do tamanho da entrada (n)

Qual é o desempenho da função `primeiro_par`?

```
1 int primeiro_par(int *v, int*n){                    custo  vezes
2     for(int i=0;i<n;i++){                             c1     x
3         if(v[i]%2==0){                                c2     x
4             return i;                                 c3     1
5         }
6     }
7     return -1;                                        c4     1
8 }
```

Desempenho em função do tamanho da entrada (n)

Qual é o desempenho da função `primeiro_par`?

```
1  int primeiro_par(int *v, int*n){                    custo  vezes
2      for(int i=0;i<n;i++){                           c1     x
3          if(v[i]%2==0){                              c2     x
4              return i;                               c3     1
5          }
6      }
7      return -1;                                      c4     1
8  }
```

x – depende da posição do primeiro número par dentro do vetor

caso médio: média dos tempos de execução de todas as entradas de tamanho n.

Requer uma distribuição de probabilidades sobre a entrada.

Se cada posição do vetor tem $\frac{1}{2}$ chances de ter um número par,então:

$$x = 1 \times \frac{1}{2} + 2 \times \frac{1}{4} + 3 \times \frac{1}{8} + \cdots + n \times \frac{1}{2^n} = \sum_{i=1}^{n} \frac{i}{2^i}$$

$$C(n) = 2\left(\sum_{i=1}^{n} \frac{i}{2^i} \right) + 1$$

Análise assintótica

Análise assintótica

Foco: verificar o crescimento do tempo de execução em função do crescimento do tamanho da entrada

Exemplo: Sejam dois algoritmos a_1 e a_2 cujo desempenho é dado, respectivamente por $C_1(n) = 10^6(n) + 10^4$ e $C_2(n) = n^2 + 1$. Qual deles é mais rápido?

Análise assintótica

Exemplo: Sejam dois algoritmos a_1 e a_2 cujo desempenho é dado, respectivamente por $C_1(n) = 10^6(n) + 10^4$ e $C_2(n) = n^2 + 1$. Qual deles é mais eficiente?

n	$C_1(n) = 10^6 n + 10^4$	$C_2(n) = n^2 + 1$
1	1.010.000	2
2	2.010.000	5
$\vdots$	$\vdots$	$\vdots$
1.000.000	$1,00000001 \times 10^{12}$	1×10^{12}
1.000.001	$1,00000101 \times 10^{12}$	$1,00000200 \times 10^{12}$
1.000.002	$1,00000201 \times 10^{12}$	$1,00000400 \times 10^{12}$

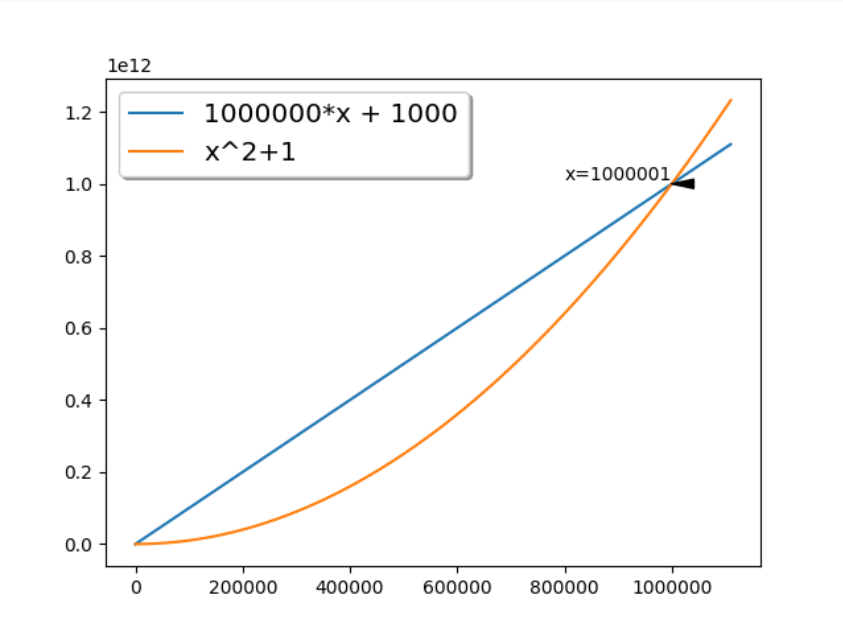
Análise assintótica

Exemplo: Sejam dois algoritmos a_1 e a_2 cujo desempenho é dado, respectivamente por $C_1(n) = 10^6(n) + 10^4$ e $C_2(n) = n^2 + 1$. Qual deles é mais eficiente?

C_2 é menor para um grande número de n

Mas, a partir de um certo n, C_2 passa a ser maior

Pode-se dizer que $C_2(n) > C_1(n)$ quando $n \to \infty$



Análise assintótica - notação O (big-O):

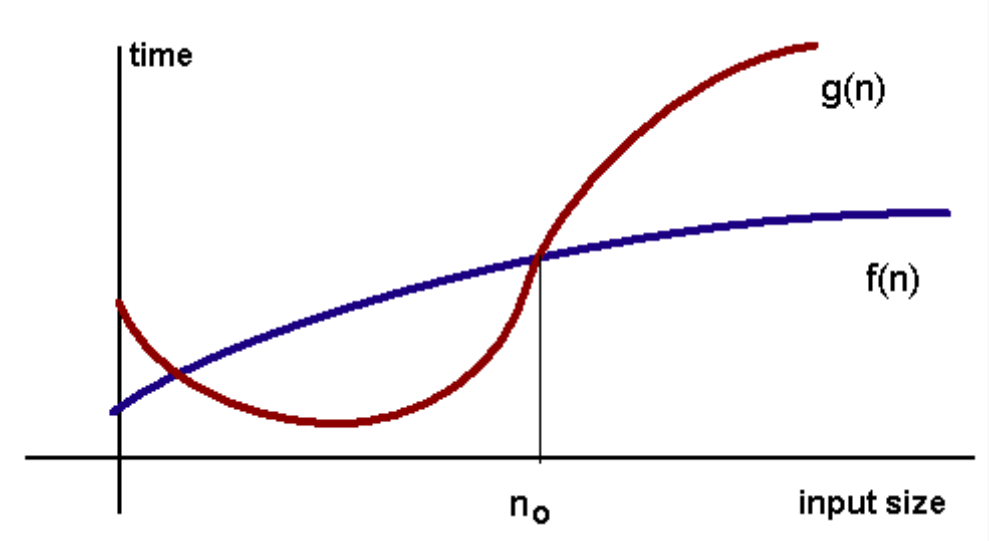
Formalmente

Sejam $f(n)$ e $g(n)$ duas funções mapeando números inteiros não negativos em números reais. Diz-se que $f(n) = \boldsymbol{O}(g(n))$ se e somente se existem constantes $c > 0$ e $n_0 \geq 1$ tais que $f(n) \leq c.g(n)$ para todo $n \geq n_0$

Informalmente

Uma função $f(n)$ é $\boldsymbol{O}(g(n))$ se existe um número n_0 a partir do qual $g(n)$ será sempre maior que $f(n)$.
ou
Uma função $f(n)$ é $\boldsymbol{O}(g(n))$ se $f(n)$ não cresce mais que $g(n)$.

Análise assintótica - notação O (big-O):



Análise assintótica - notação O (big-O):

Exemplo: Seja $f(n) = 7n - 2$ e $g(n) = n$.
Pode-se afirmar que $f(n)$ é $\boldsymbol{O}(g(n))$.

Para $c = 7$

n	$f(n) = 7n - 2$	$c.g(n) = c.n$
1	5	7
2	12	14
3	19	21
4	26	28
⋮	⋮	⋮

Análise assintótica - notação O (big-O):

Exemplo: Seja $f(n) = 10^6 n^2 + 10^4$ e $g(n) = n^2$.
Pode-se afirmar que $f(n)$ é $\boldsymbol{O}(g(n))$.
para $n \geq 1$ e $c = 1.010.001$.

Pode-se, então afirmar que $f(n)$ é $\boldsymbol{O}(n^2)$

Análise assintótica - notação O (big-O):

Na notação O, despreza-se constantes e termos de mais baixa ordem da função

- O foco está no crescimento da função ($n \to \infty$)
- Constantes e termos de baixa ordem tornam-se irrelevantes quando $n \to \infty$

Ex: $f(n) = 3n^2 + 10n + 10$

Para $n = 10$:

- $3n^2 = 73,2\%$ do resultado
- $10n = 24,2\%$ do resultado
- $10 = 2,4\%$ do resultado

Para $n = 100$:

- $3n^2 = 96,7\%$ do resultado
- $10n = 3,2\%$ do resultado
- $10 = 0,1\%$ do resultado

Análise assintótica - notação O (big-O):

Complexidade	Exemplo
$O(log_n^2)$	Busca binária
$O(n)$	Busca linear
$O(nlog_n^2)$	Quick sort (melhor caso)
$O(n^2)$	Bubble sort
$O(2^n)$	Fibonacci recursivo
$O(n!)$	Gerar todas as permutações em um conjunto

Análise assintótica - notação O (big-O):

n	log_n	n	$nlog^n$	n^2	n^3	2^n	$n!$
2	1	2	4	4	8	4	2
4	2	4	8	16	64	16	24
8	3	8	24	64	512	256	40320
16	4	16	64	256	4096	65536	
32	5	32	160	1024	32768	4294967296	
64	6	64	384	4096	262144	$1,84 \times 10^{19}$	
128	7	128	896	16384	2097152	$3,40 \times 10^{38}$	
256	8	256	2048	65536	16777216	$1,15 \times 10^{77}$	
512	9	512	4608	262144	134217728	$1,34 \times 10^{154}$	
1024	10	1024	10240	1048576	1073741824	$1,79 \times 10^{308}$	

Leituras complementares

Capítulo 1 de *Ziviani, Nivio. Projeto De Algoritmos Com Implementações Em Pascal E C 3ed.: CENGAGE, 2010. ISBN: 9788522110506*.

Capítulo 6 de *Adamson, Iain T. Data Structures and Algorithms: A First Course. Springer, 1996. ISBN: 978-1-4471-1023-1*
(disponível no acervo eletrônico da BU-UFSC)