Análise e Projeto de Algoritmos

Profa. Sheila Morais de Almeida

DAINF-UTFPR-PG

junho - 2018

Este material é preparado usando como referências os textos dos seguintes livros.

Alfred AHO, Jeffrey HOPCROFT, John ULLMAN. *The design and analysis of computer algorithms*, 1 ed., 1974.

Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST, Clifford STEIN. *Introduction to Algorithms*, 2 ed., 2001.

S. DASGUPTA, C. H. PAPADIMITRIOU, U. V. VAZIRANI. *Algorithms*, July 18, 2006.

Algoritmos

Anos 600: criação do sistema numérico decimal.

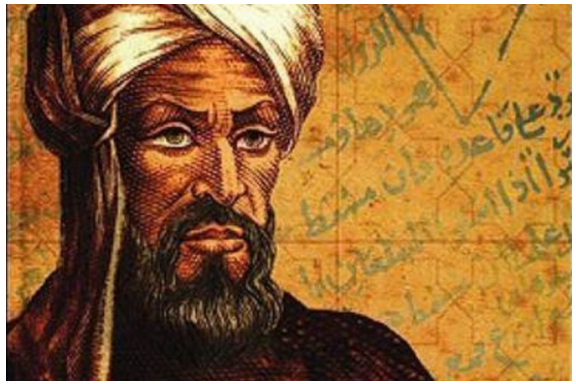
- Tornou-se mais fácil realizar cálculos.

- Propagou-se principalmente com um livro escrito por al-Khwarizmi, nos anos 800 em Bagdá.

  O livro apresentou métodos para realizar somas, multiplicações, divisões, calcular raízes quadradas e dígitos do número π.

Algoritmos

Abū ʿAbd Allāh Muḥammad ibn Mūsā al-Khwārizmī



- Apresentou a primeira solução sistemática das equações lineares e quadráticas.
- É considerado o fundador da Álgebra.
- No século XII, traduções do seu livro para latim apresentaram a notação posicional decimal para o Mundo Ocidental.
- Escreveu sobre astronomia e astrologia.

Algoritmos

Os **métodos** apresentados por al-Khawarizmi para manipulação algébrica (soma, multiplicação, divisão, etc.) são

- precisos,
- mecânicos,
- eficientes
- e corretos.

Os métodos de al-Khawarizmi são **algoritmos**.

Seu nome deu origem ao termo.

Algoritmos

Algoritmos precedem em séculos a existência de computadores.

Definição

*"**Algoritmo** é a ideia por trás dos programas de computador. É aquilo que permanece igual se o programa estiver em Pascal rodando em um supercomputador da Cray em Nova Iorque ou se estiver em Basic rodando em um Mac em Catmandu! Um algoritmo resolve um problema especificado pelo conjunto de instâncias que devem ser tratadas e por quais as propriedades que a resposta deve ter."* Steven S. Skiena

Algoritmos

Século 15: o matemático italiano Leonardo Fibonacci trabalhou no desenvolvimento e divulgação do sistema posicional decimal.

Apesar disso, Fibonacci ficou mais conhecido pela sua famosa sequência de números.

0 1 1 2 3 5 8 13 21 34 . . .

Cada número é a soma dos dois anteriores.

A Sequência de Fibonacci é formalmente definida por

$$F(n) = \begin{cases} 0 & \text{se } n = 0; \\ 1 & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{se } n > 1. \end{cases}$$

Nenhuma outra sequência de números tem sido estudada tão extensivamente, ou aplicada a mais campos: física, biologia, demografia, arte, arquitetura, música, dentre outros campos.

Algoritmos

A Sequência de Fibonacci é formalmente definida por

$$F(n) = \begin{cases} 0 & \text{se } n = 0; \\ 1 & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{se } n > 1. \end{cases}$$

Qual o valor de $F(100)$?

Podemos fazer uma algoritmo que apresenta $F(n)$ para um dado valor de n?

Formas de Representação

Algoritmos podem ser apresentados em língua natural, em linguagem de programação, como um projeto de hardware, dentre outras formas.

O importante é que a descrição seja suficientemente precisa para que o algoritmo possa ser reproduzido.

Algoritmos

Algoritmo para determinar o n-ésimo número da sequência de Fibonacci:

Fibonacci(n)

Se $n = 0$, então:
retorne 0;
Se $n = 1$, então:
retorne 1;
Se $n > 1$, então:
retorne $Fibonacci(n-1) + Fibonacci(n-2)$;

Para qualquer algoritmo, existem três perguntas que devemos nos fazer:

1) Este algoritmo está correto?

2) Quanto tempo ele demora para cada valor de n?

3) Tem como fazer melhor?

Algoritmos

Algoritmo para determinar o n-ésimo número da sequência de Fibonacci:

Fibonacci(n)

Se $n = 0$, então:
retorne 0;
Se $n = 1$, então:
retorne 1;
Se $n > 1$, então:
retorne $Fibonacci(n-1) + Fibonacci(n-2)$;

1) Este algoritmo está correto?

É exatamente a definição matemática da sequência de Fibonacci.

Algoritmos

Algoritmo para determinar o n-ésimo número da sequência de Fibonacci:

Fibonacci(n)

Se $n = 0$, então:
retorne 0;
Se $n = 1$, então:
retorne 1;
Se $n > 1$, então:
retorne $Fibonacci(n-1) + Fibonacci(n-2)$;

2) Quanto tempo ele demora para cada valor de n?

O tempo de execução para cada valor de n, denotado por $T(n)$, é limitado pelo número de instruções executadas pelo computador.

Se $n \leq 1$, o algoritmo executa no máximo 2 comparações. Então $T(n) \leq 2$.

Algoritmos

Algoritmo para determinar o n-ésimo número da sequência de Fibonacci:

Fibonacci(n)

Se $n = 0$, então:
    retorne 0;
Se $n = 1$, então:
    retorne 1;
Se $n > 1$, então:
    retorne $Fibonacci(n-1) + Fibonacci(n-2)$;

2) Quanto tempo ele demora para cada valor de n?

Se $n > 1$, o algoritmo executa $T(n) = 3 + T(n-1) + T(n-2)$ comparações.

2) Quanto tempo ele demora para cada valor de n?

Se $n > 1$, o algoritmo executa $T(n) = 3 + T(n-1) + T(n-2)$ comparações.

n	$T(n)$
0	1
1	2
2	6
3	11
4	20
5	34
6	57
7	94
8	154
9	251
10	408

Algoritmos

2) Quanto tempo ele demora para cada valor de n?

408 comparações para computar F(10).

3) Tem como fazer um algoritmo melhor?

Algoritmos

Para um mesmo problema podem existir muitos algoritmos diferentes.

Algoritmos

Uma alternativa para calcular o n-ésimo número da sequência de Fibonacci:

Fibonacci-2(n)

Criar um vetor $F[0..n]$
$F[0] \leftarrow 0$;
$F[1] \leftarrow 1$;
Para i de 2 até n faça:
    $F[i] \leftarrow F[i-1] + F[i-2]$;
retorne $F[n]$;

Algoritmos

Uma alternativa para calcular o n-ésimo número da sequência de Fibonacci:

Fibonacci-2(n)

Criar um vetor $F[0..n]$
$F[0] \leftarrow 0$;
$F[1] \leftarrow 1$;
Para i de 2 até n faça:
    $F[i] \leftarrow F[i-1] + F[i-2]$;
retorne $F[n]$;

1) Este algoritmo está correto?

2) Quanto tempo ele demora para cada valor de n?

3) Tem como fazer melhor?

Algoritmos

Uma alternativa para calcular o n-ésimo número da sequência de Fibonacci:

Fibonacci-2(n)

Criar um vetor $F[0..n]$
$F[0] \leftarrow 0$;
$F[1] \leftarrow 1$;
Para i de 2 até n faça:
    $F[i] \leftarrow F[i-1] + F[i-2]$;
retorne $F[n]$;

1) Este algoritmo está correto?

É exatamente a definição da sequência de Fibonacci.

Algoritmos

Uma alternativa para calcular o n-ésimo número da sequência de Fibonacci:

Fibonacci-2(n)

Criar um vetor $F[0..n]$
$F[0] \leftarrow 0$;
$F[1] \leftarrow 1$;
Para i de 2 até n faça:
    $F[i] \leftarrow F[i-1] + F[i-2]$;
retorne $F[n]$;

2) Quanto tempo ele demora para cada valor de n?

São no máximo 4 operações: 3 atribuições e 1 comparação para $n \leq 1$.

Mais 7 operações para cada $i \geq 2$: 4 somas/subtrações, 2 atribuições e 1 comparação.

(mais a alocação do vetor)

Algoritmos

Uma alternativa para calcular o n-ésimo número da sequência de Fibonacci:

Fibonacci-2(n)

Criar um vetor $F[0..n]$
$F[0] \leftarrow 0$;
$F[1] \leftarrow 1$;
Para i de 2 até n faça:
    $F[i] \leftarrow F[i-1] + F[i-2]$;
retorne $F[n]$;

2) Quanto tempo ele demora para cada valor de n?

São no máximo 4 operações: 3 atribuições e 1 comparação para $n \leq 1$.

Mais 7 operações para cada $i \geq 2$: 4 somas/subtrações, 2 atribuições e 1 comparação.

$$T_2(n) = 4 + 7(n-1) = 7n - 3$$

Algoritmos

Vamos comparar os dois algoritmos:

n	$T(n)$	$T_2(n)$
0	1	4
1	2	4
2	6	11
3	11	18
4	20	25
5	34	32
6	57	39
7	94	46
8	154	53
9	251	60
10	408	67

Algoritmos

Houve uma redução no número de instruções executadas pelo segundo algoritmo.

Quanto maior n, maior a diferença entre o número de instruções executadas pelo primeiro e segundo algoritmo.

Algoritmos

Uma alternativa para calcular o n-ésimo número da sequência de Fibonacci:

Fibonacci-2(n)

Criar um vetor $F[0..n]$
$F[0] \leftarrow 0$;
$F[1] \leftarrow 1$;
Para i de 2 até n faça:
    $F[i] \leftarrow F[i-1] + F[i-2]$;
retorne $F[n]$;

3) Tem como fazer melhor?

Algoritmos

Vamos economizar memória!

Fibonacci-2(n)

```
Se n = 0 então
    retorne 0;
penultimo ← 0;
ultimo ← 1;
Para i de 2 até n faça:
    tmp ← ultimo;
    ultimo ← ultimo + penultimo;
    penultimo ← tmp;
retorne ultimo;
```

2 atribuições a mais por execução do laço em troca de economizar memória alocando 3 inteiros em vez de $n + 1$ inteiros.

É melhor?

Algoritmos

Para um mesmo problema podem existir muitos algoritmos diferentes.

Conhecer seus pontos fortes e suas limitações pode ajudar a escolher qual algoritmo é melhor para cada aplicação.

Análise de Algoritmos

Para analisar um algoritmo e poder compará-lo a outros que resolvem o mesmo problema, vamos avaliar sua **corretude** e **eficiência**.

Corretude:

Considerando qualquer instância válida, o algoritmo termina? Para qualquer instância, o algoritmo apresenta a resposta esperada (de acordo com a especificação do problema)?

Eficiência

A eficiência do algoritmo é medida em termos da quantidade de recursos (memória, tempo de execução, número de processadores, acessos a disco) que o mesmo utiliza quando é executado.

Análise de Algoritmos

Em relação à eficiência, dois algoritmos desenvolvidos para resolver um mesmo problema podem ser drasticamente diferentes.

Essas diferenças podem ser muito mais significativas que diferenças de hardware ou software.

Exemplo

Considere dois algoritmos que resolvem o Problema da Ordenação: *Insertion Sort* e *Merge Sort*.

Sabemos que o número de instruções básicas do computador executadas por esses algoritmos varia conforme a quantidade de números que pecisam ser ordenados.

Veremos nessa disciplina que para ordenar uma sequência com n números, os números de instruções computacionais executadas são em média:

Insertion Sort: $c_1 n^2$, onde c_1 é uma constante que não depende de n.

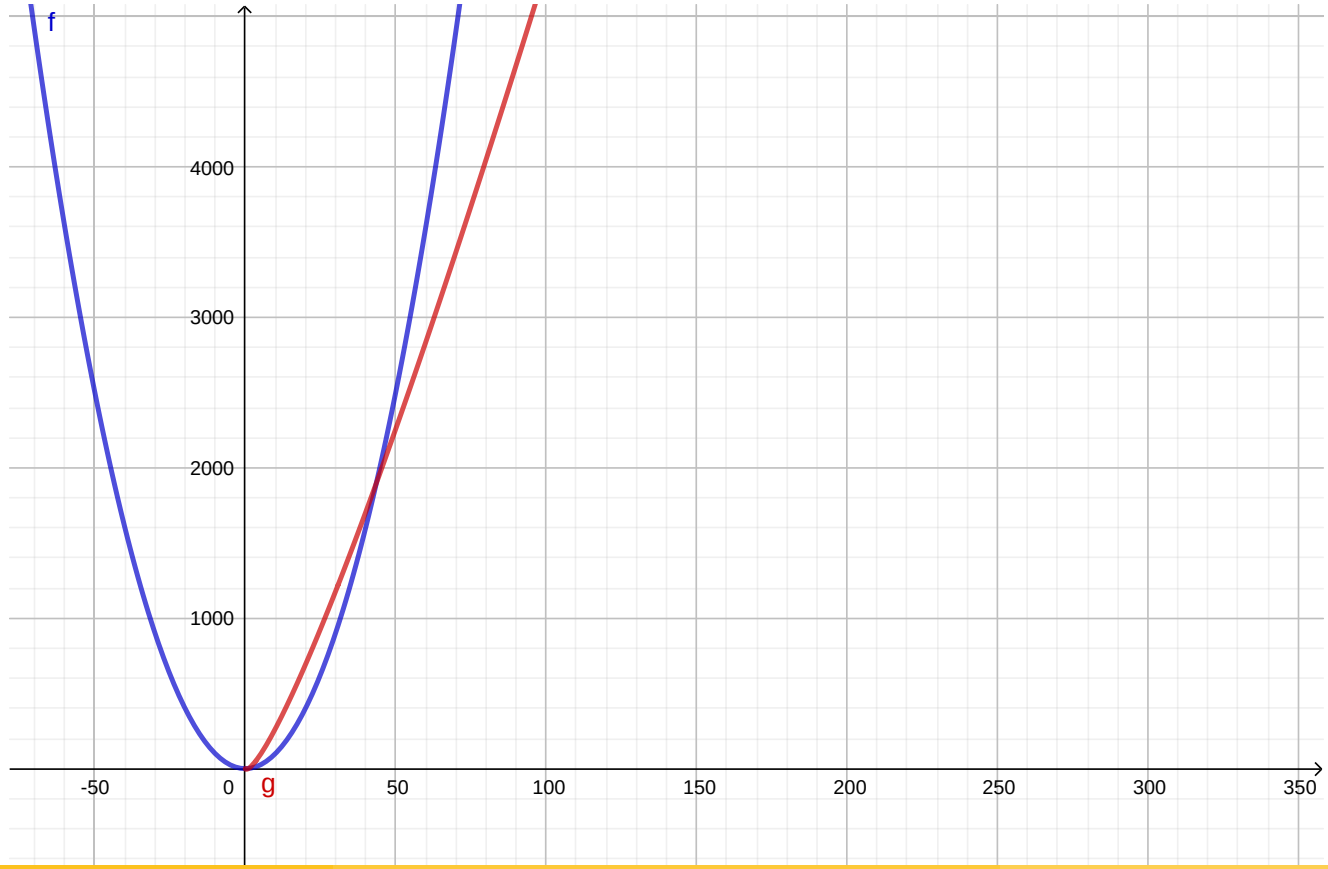
Merge Sort: $c_2 n \log n$, onde c_2 é uma constante que não depende de n.

Além disso, geralmente $c_1 < c_2$.

Exemplo

Insertion Sort: $c_1 n^2$ e *Merge Sort*: $c_2 n \log n$

Além disso, geralmente $c_1 < c_2$.



Análise de Algoritmos

Insertion Sort: $c_1 n^2$,

Merge Sort: $c_2 n \log n$,

Apesar de $c_1 < c_2$, o *Insertion Sort* só consegue ser melhor que o *Merge Sort* para instâncias pequenas (valor pequeno de n).

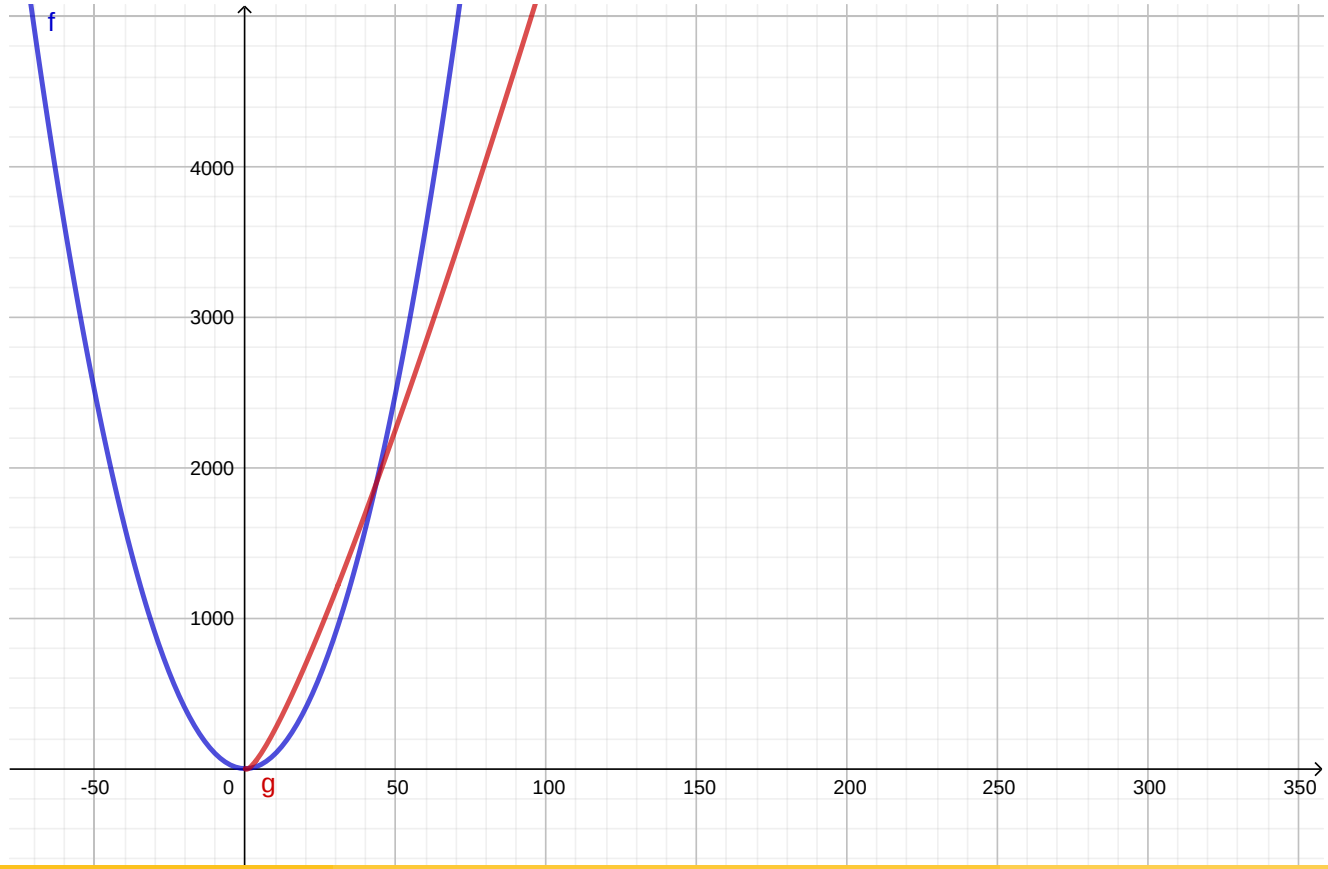
Quanto maior for n, maior é a diferença na eficiência entre esses algoritmos, destacando o desempenho do *Merge Sort*.

Interessante! Já que a constante no tempo de execução do *Insertion Sort* é menor.

Exemplo

Insertion Sort: $c_1 n^2$ e *Merge Sort*: $c_2 n \log n$

Além disso, geralmente $c_1 < c_2$.



Análise de Algoritmos

Suponha que o *Insertion Sort* (que executa $c_1 n^2$ instruções básicas do computador) e o *Merge Sort* (que executa $c_2 n \log n$ instruções básicas do computador) vão ser usados para ordenar um vetor com 1 milhão de elementos.

Mas vamos dar alguma vantagem para o *Insertion Sort*:

- computador do *Insertion Sort*: 1 bilhão de instruções por segundo;
- computador do *Merge Sort*: 10 milhões de instruções por segundo.

Ou seja, o *Insertion Sort* vai rodar em um computador 100 vezes mais rápido que o do *Merge Sort*!

Análise de Algoritmos

Vamos ordenar 1 milhão de elementos e vamos dar alguma vantagem para o *Insertion Sort*:

- o programador do *Insertion Sort* é o muito habilidoso e, além disso, o algoritmo foi implementado em linguagem de máquina. O resultado é uma constante pequena na função que determina o número de instruções computacionais que serão executadas.

  Número de instruções que serão executadas pelo *Insertion Sort*: $2n^2$.

- o programador do *Merge Sort* é mediano e, para piorar, o algoritmo foi implementado em uma linguagem de alto nível com um compilador ineficiente. O resultado foi uma constante c_2 alta.

  Número de instruções executadas pelo *Merge Sort*: $50n \log n$.

Análise de Algoritmos

Vamos ordenar 1 milhão de elementos e vamos dar alguma vantagem para o *Insertion Sort*:

- computador do *Insertion Sort*: 1 bilhão de instruções por segundo;
- computador do *Merge Sort*: 10 milhões de instruções por segundo.
- *Insertion Sort* executa um total de $2n^2$ instruções.
- *Merge Sort* executa um total de $50n \log n$ instruções.

Quem vai ser mais rápido?

Análise de Algoritmos

$$\textit{Insertion Sort}: \frac{2(10^6)^2 \text{ instruções}}{10^9 \text{ instruções por segundo}} = 2000 \text{ segundos}$$

$$\textit{Merge Sort}: \frac{50(10^6 \log(10^6)) \text{ instruções}}{10^7 \text{ instruções por segundo}} = 100 \text{ segundos}$$

Análise de Algoritmos

Vamos comparar algumas funções comuns que representam números de instruções executadas por um algoritmo para resolver problemas computacionais.

	100	1000	10^4	10^6
$\log n$	2	3	4	6
n	100	1000	10^4	10^6
$n \log n$	200	3000	$4 \cdot 10^4$	$6 \cdot 10^6$
n^2	10^4	10^6	10^8	10^{12}
$100n^2 + 15n$	$1,0015 \cdot 10^6$	$1,00015 \cdot 10^8$	$\approx 10^{10}$	$\approx 10^{14}$
2^n	$\approx 1,26 \cdot 10^{30}$	$\approx 1,07 \cdot 10^{301}$	$\approx 2 \cdot 10^{3010}$	$\approx 10^{301030}$

Análise de Algoritmos

Vejamos quanto tempo alguns algoritmos demoram para ser executados em um computador que processa 1 milhão de operações por segundo:

instruções	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
n	$0,00001$ s	$0,00002$ s	$0,00003$ s	$0,00004$ s	$0,00005$ s
n^2	$0,0001$ s	$0,0004$ s	$0,0009$ s	$0,0016$ s	$0,0025$ s
n^3	$0,001$ s	$0,008$ s	$0,027$ s	$0,064$ s	$0,125$ s
n^5	$0,1$ s	$3,2$ s	$24,3$ s	$1,7$ min	$5,2$ min
2^n	$0,001$ s	$1,04$ s	$17,9$ min	$12,7$ dias	$35,7$ anos
3^n	$0,059$ s	58 min	$6,5$ anos	3.855 séc	10^8 séc

Análise de Algoritmos

Para desenvolver algoritmos eficientes é preciso preocupar-se com o número de instruções que são executadas para resolver o problema.

Objetivos

- conhecer técnicas de projeto de algoritmos;
- saber calcular o número de instruções que serão executadas pelo algoritmo;
- saber identificar se o algoritmo projetado é eficiente e utiliza quantidade adequada de recursos para resolver o problema.