01 - Introdução à Análise de Algoritmos

Análise e Projeto de Algoritmos

Sheila Morais de Almeida

Universidade Tecnológica Federal do Paraná

17 de abril de 2023

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.

Udi MAMBER, *Introduction to Algorithms: A Creative Approach*, 1989.

Algoritmos

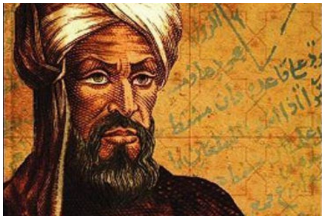
Anos 600: criação do sistema numérico decimal.

- 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.

Algoritmos

A Sequência de Fibonacci é formalmente definida por

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

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 ações executadas para a conclusão da tarefa.

Algoritmos

Para fazer uma previsão do tempo necessário para a execução de um algoritmo, considerando uma determinada instância do problema, devemos:

- Identificar o conjunto Q dos tipos de instruções que devem ser executadas.
- Determinar $tempo(q)$, o tempo necessário para a execução de cada tipo de instrução $q \in Q$.
- Determinar $qtd(q)$: a quantidade de vezes que a instrução de cada tipo $q \in Q$ é executada.
- Calcular o tempo total de execução:

$$T(n) = \sum_{\forall q \in Q} (qtd(q) \cdot tempo(q))$$

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?

Neste exemplo, as instruções são de três tipos:

- comparações
- somas
- subtrações

Se $n \leq 1$, o algoritmo executa no máximo 2 comparações. Então $T(n) \leq 2 \cdot \text{tempo(comparação)}$.

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 $n > 1$, são 3 comparações, duas subtrações e uma adição, além das chamadas recursivas. Então:

$T(n) = 3 \cdot \text{tempo(comparação)} + 2 \cdot \text{tempo(subtração)} +$
$1 \cdot \text{tempo(adição)} + T(n-1) + T(n-2).$

Algoritmos

2) Quanto tempo Fibonacci(n) demora para cada valor de n?

Suponha tempo(q) $= 1\mu s$, para qualquer tipo de operação q. Se $n > 1$, o algoritmo demora $T(n) = 6 + T(n-1) + T(n-2)\mu s$.

n	$T(n)$(em μs)
0	1
1	2
2	9
3	17
4	32
5	55
6	93
7	154
8	253

Algoritmos

2) Quanto tempo ele demora para $n = 8$?

$253\mu s$ para computar Fibonacci(8).

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 $n = 8$?

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 $n = 8$?

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

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

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 $n = 8$?
São no máximo 3 atribuições e 1 comparação para $n \leq 1$.

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

$T_2(n) = 4 + 7(n-1) = 7n - 3\mu s$.
Supondo tempo(q) $= 1\mu s, \forall q$, tem-se: $T_2(8) = 4 + 7(7) = 53\mu s$

Algoritmos

Vamos comparar os dois algoritmos:

n	$T(n)$	$T_2(n)$
0	1	4
1	2	4
2	9	11
3	17	18
4	32	25
5	55	32
6	93	39
7	154	46
8	253	53

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.

Análise de Algoritmos

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.

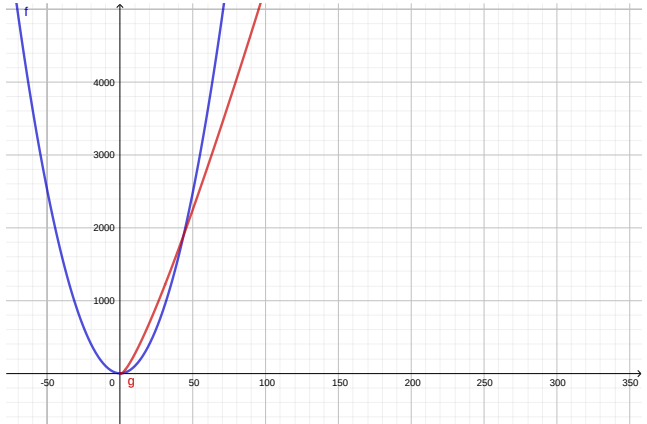
Além disso, geralmente $c_1 < c_2$.

Análise de Algoritmos

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*.

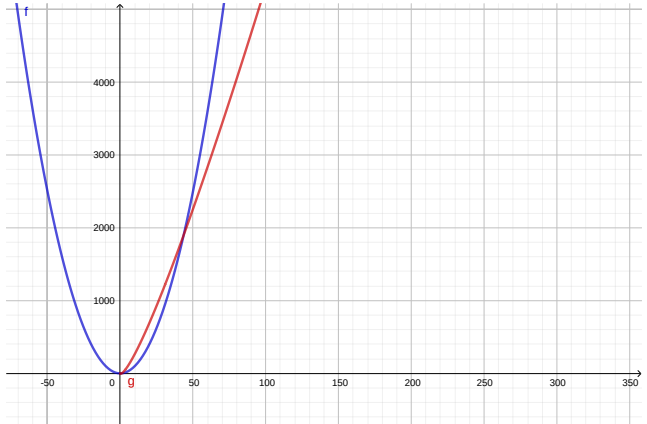
Interessante! Já que a constante no tempo de execução do *Insertion Sort* é menor.

Análise de Algoritmos

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

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

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	$0,0001$ 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.

Análise de Algoritmos

Estamos interessados em avaliar a **eficiência** de um algoritmo.

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.

Na maioria dos casos, vamos medir a eficiência em **tempo de execução**.

Modelo Computacional RAM

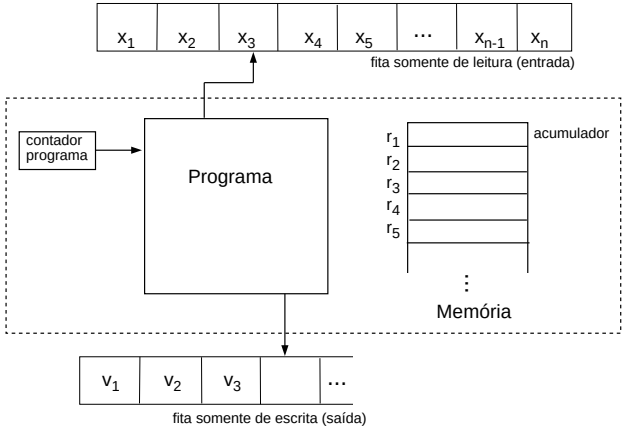
A análise de um algoritmo depende do modelo computacional adotado.

É de acordo com o modelo computacional que se define quais são os recursos disponíveis e quanto custam.

Modelo RAM - *Random Access Machine*

- Simula máquinas convencionais.
- Um único processador que executa instruções sequencialmente.
- Operações aritméticas básicas (somas, subtrações, multiplicações e divisões), atribuições e comprações são feitas em tempo constante.

Modelo Computacional RAM



Fonte: **A. Aho; J. Hopcroft; J. Ullman**. *The Design and Analysis of Computer Algorithms*, 1974

Modelo Computacional RAM

No modelo RAM o programa não é armazenado na memória, então o espaço que ele ocupa não é contabilizado no gasto de memória.

Instruções existentes em computadores reais podem ser incorporadas ao modelo RAM sem alterar a ordem de grandeza da complexidade dos problemas[1].

[1] **A. Aho; J. Hopcroft; J. Ullman**. *The Design and Analysis of Computer Algorithms*, 1974

Análise de Algoritmos

Para fazer uma previsão do tempo necessário para a execução de um algoritmo em determinado modelo computacional, considerando uma determinada instância do problema, devemos:

- Identificar o conjunto Q das instruções computacionais básicas do modelo.
- Determinar $t(q)$, o tempo necessário para a execução de cada instrução $q \in Q$.
- Determinar $f(q)$, a quantidade de vezes que cada instrução $q \in Q$ é executada.
- Calcular o tempo total de execução:

$$\sum_{\forall q \in Q} f(q) \cdot t(q)$$

Análise de Algoritmos

Exemplo: análise do Algoritmo de Ordenação por Inserção.

Quantas operações de comparação são executadas no *Insertion Sort*?

Algoritmo Ordenação por Inserção
Entrada: $v[1..n]$; n;
    Para i de 2 a n **faça:**
        $j \leftarrow i - 1$
        Enquanto $j > 0$ && $v[i] < v[j]$ **faça:**
            $j \leftarrow j - 1$
        $t \leftarrow v[i]$
        Para k de $i - 1$ a $j + 1$ **faça:**
            $v[k + 1] \leftarrow v[k]$
        $v[j + 1] \leftarrow t$

Análise de Algoritmos

Exemplo: análise do Algoritmo de Ordenação por Inserção.

Quantas operações de comparação são executadas no *Insertion Sort*?
Depende da instância.

Algoritmo Ordenação por Inserção
Entrada: $v[1..n]$; n;
    Para i de 2 a n **faça:**
        $j \leftarrow i - 1$
        Enquanto $j > 0$ && $v[i] < v[j]$ **faça:**
            $j \leftarrow j - 1$
        $t \leftarrow v[i]$
        Para k de $i - 1$ a $j + 1$ **faça:**
            $v[k + 1] \leftarrow v[k]$
        $v[j + 1] \leftarrow t$

Análise de Algoritmos

Podemos considerar todas as instâncias possíveis!

Seria uma tarefa bastante árdua! Quantas entradas possíveis existem?

Vamos usar uma medida da entrada, que chamamos de **tamanho da entrada**.

- A análise dos algoritmos será em função do tamanho da entrada.
- O tamanho da entrada é normalmente denotado por n.
- Em geral, o tamanho da entrada é uma medida da quantidade de memória necessária para armazenar a entrada.

Tamanho da entrada

- Deve-se considerar as unidades básicas das estruturas de dados que serão manipuladas pelo algoritmo:
  - em uma ordenação em um vetor, o número de elementos;
  - no cálculo com números muito grandes (para astronomia, por exemplo), a quantidade de dígitos dos números;
  - em um grafo, a quantidade de vértices e arestas.

Análise de Algoritmos

O algoritmo pode não se comportar exatamente da mesma forma para todas as instâncias de tamanho n.

Nós teremos que escolher, dentre todas as instâncias possíveis, qual queremos usar como indicativo do tempo de execução do algoritmo.

Escolha das instâncias para análise

Normalmente, considera-se:

- a entrada do melhor caso, aquela que faz com que o algoritmo execute o menor número de instruções para dar a resposta;
- a entrada de caso médio, aquela que exige um tempo de execução que é aproximadamente a média dos tempos de execução de todas as possíveis entradas; ou
- a entrada de pior caso, aquela que faz o algoritmo demorar mais (executar mais instruções) para dar a resposta.

Sobre a entrada de melhor caso

A análise com entrada de melhor caso geralmente não representa o que ocorre com o tempo de execução de um algoritmo na maior parte dos casos.

A maioria dos problemas tem uma instância para a qual a sua solução é trivial.

Sobre a entrada de caso médio

Existem alguns casos famosos em que a análise realizada com entradas de caso médio apresenta resultados bem melhores que a análise com entradas de pior caso.

Alguns problemas são intratáveis no pior caso, mas as entradas que explicitam esse comportamento podem raramente ocorrer na prática.

- A complexidade de caso médio pode ser uma medida mais precisa da performance desses algoritmos.

Sobre a entrada de caso médio

Poderia ser uma boa escolha! Mas...

O que é uma entrada do caso médio? Nem sempre é claro o que é uma entrada de caso médio.

Que parâmetros serão usados para se tirar a média?

Se não tomarmos cuidado, podemos considerar como entrada de caso médio um tipo de entrada que nunca ocorre na prática.

É difícil avaliar o desempenho de entradas de caso médio. Geralmente, exigem maior habilidade matemática.

Sobre a entrada de pior caso

A análise com entrada de pior caso fornece um **limite superior** para o tempo de execução do algoritmo. Conhecer esse limite é ter uma garantia de que o algoritmo não vai demorar mais do que essa medida.

Em alguns casos, o **pior caso ocorre com frequência**: na busca em um banco de dados, o pior caso é quando o dado não está no banco.

Sobre a entrada de pior caso

Frequentemente, os resultados da análise com entradas de pior caso **são muito próximos dos obtidos com entradas de caso médio** ou através de observações experimentais.

Mesmo quando a análise com entradas de pior caso tem resultados diferentes da análise com entradas de caso médio, **o algoritmo que tem o melhor desempenho no pior caso também tem desempenho muito bom com as demais instâncias**.

Análise de Algoritmos

Para o Algoritmo de Ordenação por Inserção, apresente uma instância

- de melhor caso;
- de pior caso;
- e de caso médio.

Algoritmo Ordenação por Inserção
Entrada: $v[1..n]$; n;
    Para i de 2 a n **faça:**
        $j \leftarrow i - 1$
        Enquanto $j > 0$ && $v[i] < v[j]$ **faça:**
            $j \leftarrow j - 1$
        $t \leftarrow v[i]$
        Para k de $i - 1$ a $j + 1$ **faça:**
            $v[k + 1] \leftarrow v[k]$
        $v[j + 1] \leftarrow t$

Análise de Algoritmos

Para o Algoritmo de Ordenação por Inserção, apresente uma instância

- de melhor caso: vetor em ordem crescente.
- de pior caso: vetor em ordem decrescente.
- e de caso médio:
  - quando $v[i]$ é considerado, tem a mesma chance de ser inserido em qualquer posição de $v[1]$ a $v[i]$;
  - estatisticamente, a chance de $v[i]$ ser inserido na posição $j \in [1, i]$ é $1/i$;
  - então o tempo médio de execução do algoritmo nesse caso, é a média de quantas instruções são executadas para se inserir $v[i]$ na posição j, para cada $j \in [1, i]$.

Análise de Algoritmos

Para o Algoritmo de Ordenação por Inserção, apresente uma instância

- de melhor caso: vetor em ordem crescente.
- de pior caso: vetor em ordem decrescente.
- e de caso médio:
  - quando $v[i]$ é considerado, tem a mesma chance de ser inserido em qualquer posição de $v[1]$ a $v[i]$;
  - estatisticamente, a chance de $v[i]$ ser inserido na posição $j \in [1, i]$ é $1/i$;
  - então o tempo médio de execução do algoritmo nesse caso, é a média de quantas instruções são executadas para se inserir $v[i]$ na posição j, para cada $j \in [1, i]$.

Análise de complexidade de tempo no pior caso

Algoritmo Ordenação por Inserção
Entrada: $v[1..n]$; n;
    Para i de 2 a n **faça:**
        $j \leftarrow i - 1$
        Enquanto $j > 0$ && $v[i] < v[j]$ **faça:**
            $j \leftarrow j - 1$
        $t \leftarrow v[i]$
        Para k de $i - 1$ a $j + 1$ **faça:**
            $v[k + 1] \leftarrow v[k]$
        $v[j + 1] \leftarrow t$

Análise de complexidade de tempo no pior caso

Pergunta: Quantas instruções básicas do modelo computacional RAM (operações aritméticas básicas, atribuições e comparações) são executadas pelo Algoritmo de Ordenação por Inserção, considerando uma entrada de tamanho n de pior caso?

Análise de complexidade de tempo no pior caso

Algoritmo Ordenação por Inserção
Entrada: $v[1..n]$; n;

Para i de 2 a n **faça:** $2 + 3(n-1) = 3n - 1$
- $j \leftarrow i - 1$ $2(n-1) = 2n - 2$
- **Enquanto** $j > 0$ && $v[i] < v[j]$ **faça:** $2\sum_{c=1}^{n-1} c + (n-1)^*$
  - $j \leftarrow j - 1$ $2\sum_{c=1}^{n-1} c = n^2 - n$
- $t \leftarrow v[i]$ $n - 1$
- **Para** k de $i - 1$ a $j + 1$ **faça:** $4\frac{n(n-1)}{2} + 4(n-1)^\dagger$
  - $v[k+1] \leftarrow v[k]$ $2\frac{n(n-1)}{2} = n^2 - n$
- $v[j+1] \leftarrow t$ $2(n-1) = 2n - 2$

$^*2\sum_{c=1}^{n-1} c + (n-1) = n(n-1) + (n-1) = (n+1)(n-1) = n^2 - 1$

$^\dagger 4\frac{n(n-1)}{2} + 4(n-1) = 2n^2 + 2n - 4$

Análise de complexidade de tempo no pior caso

Algoritmo Ordenação por Inserção
Entrada: $v[1..n]$; n;
    Para i de 2 a n **faça:** $2 + 3(n-1) = 3n - 1$
        $j \leftarrow i - 1$ $2(n-1) = 2n - 2$
        Enquanto $j > 0$ && $v[i] < v[j]$ **faça:** $n^2 - 1$
            $j \leftarrow j - 1$ $2\sum_{c=1}^{n-1} c = n^2 - n$
        $t \leftarrow v[i]$ $n - 1$
        Para k de $i - 1$ a $j + 1$ **faça:** $2n^2 + 2n - 4$
            $v[k+1] \leftarrow v[k]$ $2\frac{n(n-1)}{2} = n^2 - n$
        $v[j+1] \leftarrow t$ $2(n-1) = 2n - 2$

Total: $5n^2 + 8n - 11$

Análise de complexidade de tempo no caso médio

Algoritmo Ordenação por Inserção
Entrada: $v[1..n]$; n;

Para i de 2 a n **faça:** $2 + 3(n-1) = 3n - 1$

$j \leftarrow i - 1$ $2(n-1) = 2n - 2$

Enquanto $j > 0$ && $v[i] < v[j]$ **faça:** $\sum_{i=2}^{n}[\sum_{j=0}^{i-1}(\sum_{c=j}^{i-1} 2)/i]$

$j \leftarrow j - 1$ $\sum_{i=2}^{n}[\sum_{j=0}^{i-1}(\sum_{c=j+1}^{i-1} 2)/i]$

$t \leftarrow v[i]$ $n - 1$

Para k de $i - 1$ a $j + 1$ **faça:** $\sum_{i=2}^{n}[\sum_{j=0}^{i-1}(\sum_{k=j}^{i-1} 4)/i]$

$v[k+1] \leftarrow v[k]$ $\sum_{i=2}^{n}[\sum_{j=0}^{i-1}(\sum_{k=j+1}^{i-1} 2)/i]$

$v[j+1] \leftarrow t$ $2(n-1) = 2n - 2$

Análise de complexidade de tempo no caso médio

Algoritmo Ordenação por Inserção
Entrada: $v[1..n]$; n;
    Para i de 2 a n **faça:** $2 + 3(n-1) = 3n - 1$
        $j \leftarrow i - 1$ $2(n-1) = 2n - 2$
        Enquanto $j > 0$ && $v[i] < v[j]$ **faça:** $(n^2 + 3n - 4)/2$
            $j \leftarrow j - 1$ $(n^2 + 2n - 3)/2$
        $t \leftarrow v[i]$ $n - 1$
        Para k de $i - 1$ a $j + 1$ **faça:** $(n^2 + 3n - 4)/2$
            $v[k + 1] \leftarrow v[k]$ $(n^2 + 2n - 3)/2$
        $v[j + 1] \leftarrow t$ $2(n-1) = 2n - 2$

Total: $2n^2 + 13n - 13$

Uma nota sobre a complexidade de espaço

A Complexidade de Espaço é medida em função da quantidade de memória auxiliar e total necessária para a execução do algoritmo.

Na memória auxiliar não são considerados os espaços necessários para:

- o próprio programa;
- a entrada;
- e a saída.

Uma nota sobre a complexidade de espaço

- O armazenamento do próprio programa é desconsiderado pois é independente do tamanho da entrada.

- Os armazenamentos da entrada e da saída não são considerados pois, na comparação de diferentes algoritmos que resolvem o mesmo problema, todos ocupam a mesma quantidade de memória para armazenamento da entrada e da saída.

Uma nota sobre a complexidade de espaço

Da mesma forma que na complexidade de tempo, pode-se fazer uma análise com a entrada de tamanho n de melhor caso, caso médio ou pior caso.

Um algoritmo A_1 que necessite de $f_1(n) = 10n$ unidades de memória é um algoritmo com complexidade de espaço linear em função do tamanho da entrada.

Um algoritmo A_2 que necessite de $f_2(n) = 50$ unidades de memória (independente do valor de n) é um algoritmo com complexidade de espaço constante em função do tamanho da entrada.

Observe que se $n > 5$ o algoritmo A_2 utiliza menos memória que o algoritmo A_1 para resolver o problema.

Uma nota sobre a complexidade de espaço

Considere o Algoritmo de Ordenação por Inserção:

Algoritmo Ordenação por Inserção
Entrada: $v[1..n]$; n;
    Para i de 2 a n **faça:**
        $j \leftarrow i - 1$
        Enquanto $j > 0$ && $v[i] < v[j]$ **faça:**
            $j \leftarrow j - 1$
        $t \leftarrow v[i]$
        Para k de $i - 1$ a $j + 1$ **faça:**
            $v[k + 1] \leftarrow v[k]$
        $v[j + 1] \leftarrow t$

Uma nota sobre a complexidade de espaço

No Algoritmo de Ordenação por Inserção, independente de a entrada ser de melhor caso, caso médio ou pior caso, o espaço utilizado para memória auxiliar é:

- 1 inteiro, para armazenar a variável i;
- 1 inteiro, para armazenar a variável j;
- 1 inteiro, para armazenar a variável k;
- e 1 inteiro, para armazenar a variável t.

Logo, o algoritmo de ordenação por inserção utiliza uma quantidade de memóra auxiliar constante (4 vezes o tamanho de um inteiro), independente do tamanho da entrada.

A função que descreve a quantidade de memória auxiliar do algoritmo de ordenação por inserção é $f(n) = 4$ e, portanto, é constante em relação ao tamanho da entrada.

Uma nota sobre a complexidade de espaço

No Algoritmo de Ordenação por Inserção, independente de a entrada ser de melhor caso, caso médio ou pior caso, o espaço total utilizado da memória é:

- 1 inteiro, para armazenar a variável i;
- 1 inteiro, para armazenar a variável j;
- 1 inteiro, para armazenar a variável k;
- 1 inteiro, para armazenar a variável t;
- e $n+1$ inteiros para armazenar a quantidade de números da sequência e quais são esses números.

O algoritmo de ordenação por inserção utiliza uma quantidade de memóra total linear ($(n+5)$ vezes o tamanho de um inteiro), onde n é o tamanho da entrada.

A função que descreve a quantidade total de memória usada pelo algoritmo de ordenação por inserção é $g(n) = n+5$, que é linear em função do tamanho da entrada.