Projeto e Análise de Algoritmos

Hamilton José Brumatto

Bacharelado em Ciência da Computação - UESC

4 de abril de 2016

O Melhor e o Pior Caso
O Caso Médio

Melhor e Pior Casos

Considere o seguinte problema computacional:

Problema da soma:

Dada uma seqüência de números, indicar a soma destes números.

Qual o algoritmo que resolve este problema?
Este algoritmo está correto?
A eficiência depende da entrada?

Proposta de algoritmo:

Entrada: Vetor de elementos inteiros
Saída: Soma dos elementos no vetor

Algoritmo $\mathrm{SOMA}(\mathrm{A})$
    soma $\leftarrow 0$
    para $i \leftarrow 1$ **até** *comprimento*$[A]$ **faça**
        soma $\leftarrow$ *soma* $+ A[i]$
    Retorna *soma*

Invariante: Antes da iteração i, *soma* contém a soma dos elementos do subarranjo $A[1..i-1]$ O algoritmo está correto?
Melhor caso: $\Theta(n)$ (por quê)?
Pior caso: $\Theta(n)$ (pode ser o mesmo)?

Proposta de algoritmo:

Entrada: Vetor de elementos inteiros
Saída: Soma dos elementos no vetor

Algoritmo $\mathrm{SOMA}(\mathrm{A})$
  $soma \leftarrow 0$
  para $i \leftarrow 1$ **até** $comprimento[A]$ **faça**
    $soma \leftarrow soma + A[i]$
  Retorna $soma$

Invariante: Antes da iteração i, $soma$ contém a soma dos elementos do subarranjo $A[1..i-1]$ O algoritmo está correto?
Melhor caso: $\Theta(n)$ (por quê)?
Pior caso: $\Theta(n)$ (pode ser o mesmo)?

Proposta de algoritmo:

Entrada: Vetor de elementos inteiros
Saída: Soma dos elementos no vetor

Algoritmo $\mathrm{SOMA}(A)$
  $soma \leftarrow 0$
  para $i \leftarrow 1$ **até** $comprimento[A]$ **faça**
    $soma \leftarrow soma + A[i]$
  Retorna $soma$

Invariante: Antes da iteração i, $soma$ contém a soma dos elementos do subarranjo $A[1..i-1]$ O algoritmo está correto?
Melhor caso: $\Theta(n)$ (por quê)?
Pior caso: $\Theta(n)$ (pode ser o mesmo)?

Proposta de algoritmo:

Entrada: Vetor de elementos inteiros
Saída: Soma dos elementos no vetor

Algoritmo $\mathrm{SOMA(A)}$
  soma $\leftarrow 0$
  para $i \leftarrow 1$ **até** *comprimento*$[A]$ **faça**
    soma $\leftarrow$ *soma* $+ A[i]$
  Retorna *soma*

Invariante: Antes da iteração i, *soma* contém a soma dos elementos do subarranjo $A[1..i-1]$ O algoritmo está correto?
Melhor caso: $\Theta(n)$ (por quê)?
Pior caso: $\Theta(n)$ (pode ser o mesmo)?

Proposta de algoritmo:

Entrada: Vetor de elementos inteiros
Saída: Soma dos elementos no vetor

Algoritmo $\mathrm{SOMA}(\mathrm{A})$
  soma $\leftarrow 0$
  para $i \leftarrow 1$ **até** *comprimento*$[A]$ **faça**
    soma $\leftarrow$ *soma* $+ A[i]$
  Retorna *soma*

Invariante: Antes da iteração i, *soma* contém a soma dos elementos do subarranjo $A[1..i-1]$ O algoritmo está correto?
Melhor caso: $\Theta(n)$ (por quê)?
Pior caso: $\Theta(n)$ (pode ser o mesmo)?

Quando a eficiência depende da entrada

Considere o seguinte problema computacional:

Problema da soma:

Dada uma seqüência de números não negativos, indicar se a soma destes números é maior que um valor x.

Qual o algoritmo que resolve este problema?
Este algoritmo está correto?
A eficiência depende da entrada?

Proposta de algoritmo:

Entrada: Um vetor de elementos e um valor
Saída: Soma dos elementos do vetor $>$ valor ?

Algoritmo SOMA_MAIOR(A,x)
  soma $\leftarrow 0$
  para $i \leftarrow 1$ **até** $comprimento[A]$ **faça**
    soma $\leftarrow soma + A[i]$
  se $soma > x$ **então**
    Retorna *TRUE*
  senão
    Retorna *FALSE*

O algoritmo está correto?
Qual a eficiência?
Dá para fazer melhor?

Proposta de algoritmo:

Entrada: Um vetor de elementos e um valor
Saída: Soma dos elementos do vetor > valor ?

Algoritmo SOMA_MAIOR(A,x)
- *soma* ← 0
- **para** *i* ← 1 **até** *comprimento*[*A*] **faça**
  - *soma* ← *soma* + *A*[*i*]
- **se** *soma* > *x* **então**
  - **Retorna** *TRUE*
- **senão**
  - **Retorna** *FALSE*

O algoritmo está correto?
Qual a eficiência?
Dá para fazer melhor?

Algoritmo melhor:

Entrada: Um vetor de elementos e um valor
Saída: Soma dos elementos do vetor $>$ valor ?

Algoritmo SOMA_MAIOR(A,x)
- $soma \leftarrow A[1]$
- $i \leftarrow 2$
- **enquanto** $soma \leqslant x$ **e** $i \leqslant comprimento[A]$ **faça**
  - $soma \leftarrow soma + A[i]$
  - $i \leftarrow i + 1$
- **se** $soma > x$ **então**
  - **Retorna** *TRUE*
- **senão**
  - **Retorna** *FALSE*

O invariante do loop é outro: Antes da iteração i

- *soma* contém a soma dos valores no subarranjo $A[1..i-1]$
- a soma dos valores no subarranjo $A[1..i-2]$ não é maior que x

Mudou em algo a eficiência?
Pior Caso: $\Theta(n)$, Melhor Caso: $\Theta(1)$!!

Algoritmo melhor:

Entrada: Um vetor de elementos e um valor
Saída: Soma dos elementos do vetor $>$ valor ?

Algoritmo SOMA_MAIOR(A,x)
    $soma \leftarrow A[1]$
    $i \leftarrow 2$
    enquanto $soma \leqslant x$ **e** $i \leqslant comprimento[A]$ **faça**
        $soma \leftarrow soma + A[i]$
        $i \leftarrow i + 1$
    se $soma > x$ **então**
        Retorna *TRUE*
    senão
        Retorna *FALSE*

O invariante do loop é outro: Antes da iteração i

- $soma$ contém a soma dos valores no subarranjo $A[1..i-1]$
- a soma dos valores no subarranjo $A[1..i-2]$ não é maior que x

Mudou em algo a eficiência?
Pior Caso: $\Theta(n)$, Melhor Caso: $\Theta(1)$!!

Algoritmo melhor:

Entrada: Um vetor de elementos e um valor
Saída: Soma dos elementos do vetor > valor ?

```
Algoritmo SOMA_MAIOR(A,x)
    soma ← A[1]
    i ← 2
    enquanto soma ⩽ x e i ⩽ comprimento[A] faça
        soma ← soma + A[i]
        i ← i + 1
    se soma > x então
        Retorna TRUE
    senão
        Retorna FALSE
```

O invariante do loop é outro: Antes da iteração i

- *soma* contém a soma dos valores no subarranjo $A[1..i-1]$
- a soma dos valores no subarranjo $A[1..i-2]$ não é maior que x

Mudou em algo a eficiência?
Pior Caso: $\Theta(n)$, Melhor Caso: $\Theta(1)$!!

Situação no caso médio

O cálculo do caso médio é um pouco mais complexo, pois teríamos de considerar todas as possíveis entradas do sistema. Uma forma de estudar o caso médio é considerar que todas as entradas possuem igual probabilidade, outra forma seria trabalhar como geração aleatória de entradas e fazer uma análise probabilística.

Se todas entradas têm a mesma probabilidade, podemos considerar o caso médio como:

$$T_{\text{médio}}(n) = \frac{1}{n}\sum_{i=1}^{n} T_i(n)$$

Situação no caso médio

$$
\begin{aligned}
T_{\text{médio}}(n) &= \frac{1}{n}\sum_{i=1}^{n} T_i(n) \\
T_{\text{médio}}(n) &= \frac{1}{n}\sum_{i=1}^{n} i \qquad \text{Aproximação simplista} \\
T_{\text{médio}}(n) &= \frac{1}{n}\frac{n(n+1)}{2} = \frac{n+1}{2} \\
T_{\text{médio}}(n) &= \Theta(n)
\end{aligned}
$$

Este cálculo está correto?

Situação no caso médio

$$\begin{aligned}
T_{\text{médio}}(n) &= \frac{1}{n}\sum_{i=1}^{n} T_i(n) \\
T_{\text{médio}}(n) &= \frac{1}{n}\sum_{i=1}^{n} i \qquad \text{Aproximação simplista} \\
T_{\text{médio}}(n) &= \frac{1}{n}\frac{n(n+1)}{2} = \frac{n+1}{2} \\
T_{\text{médio}}(n) &= \Theta(n)
\end{aligned}$$

Este cálculo está correto?

Cota inferior

Retomemos o problema computacional da soma de uma seqüência de elementos. Encontramos um algoritmos que resolve o problema em $T(n) = \Theta(n)$. Isto é bom?

Para conseguirmos uma soma, precisamos do valor de cada elemento, temos que visitar cada um dos n elementos, logo a solução deste problema não pode ter um algoritmo com tempo melhor que $\Theta(n)$.

Se um algoritmo *no pior caso* tem eficiência que representa a cota inferior do problema, então ele tem ótima eficiência.

Mas será ele o mais eficiente, o que executa em tempo menor?

Cota inferior

Retomemos o problema computacional da soma de uma seqüência de elementos. Encontramos um algoritmos que resolve o problema em $T(n) = \Theta(n)$. Isto é bom?

Para conseguirmos uma soma, precisamos do valor de cada elemento, temos que visitar cada um dos n elementos, logo a solução deste problema não pode ter um algoritmo com tempo melhor que $\Theta(n)$.

Se um algoritmo *no pior caso* tem eficiência que representa a cota inferior do problema, então ele tem ótima eficiência.

Mas será ele o mais eficiente, o que executa em tempo menor?

Problema MinMax

Achar o valor mínimo e o valor máximo em uma sequência

Entrada: Uma sequência de números

Saída: Maior valor e Menor valor da sequência

Algoritmo MINMAX(A,n)

$min \leftarrow max \leftarrow A[1]$

para $j \leftarrow 2$ **até** n **faça**

  se $A[j] < min$ **então**

    $min \leftarrow A[i]$

  se $A[j] > max$ **então**

    $max \leftarrow A[i]$

Retorna min, max

São $2(n-1)$ comparações: $T(n) = 2n - 2 = \Theta(n)$

Dá para fazer melhor?

Problema MinMax

Achar o valor mínimo e o valor máximo em uma sequência

Entrada: Uma sequência de números

Saída: Maior valor e Menor valor da sequência

Algoritmo $\textsc{MinMax}$(A,n)

$min \leftarrow max \leftarrow A[1]$

para $j \leftarrow 2$ **até** n **faça**

  se $A[j] < min$ **então**

    $min \leftarrow A[i]$

  se $A[j] > max$ **então**

    $max \leftarrow A[i]$

Retorna min, max

São $2(n-1)$ comparações: $T(n) = 2n - 2 = \Theta(n)$

Dá para fazer melhor?

Problema MinMax

Achar o valor mínimo e o valor máximo em uma sequência

Entrada: Uma sequência de números

Saída: Maior valor e Menor valor da sequência

Algoritmo MinMax(A,n)
- $min \leftarrow max \leftarrow A[1]$
- **para** $j \leftarrow 2$ **até** n **faça**
  - **se** $A[j] < min$ **então**
    - $min \leftarrow A[i]$
  - **se** $A[j] > max$ **então**
    - $max \leftarrow A[i]$
- **Retorna** min, max

São $2(n-1)$ comparações: $T(n) = 2n - 2 = \Theta(n)$

Dá para fazer melhor?

Problema MinMax

- Processe os elementos aos pares, para cada par, compare o menor com o valor mínimo atual e o maior com o valor máximo atual. São 3 comparações para cada 2 elementos.
- Se n for ímpar: iniciar os valores *min* e *max* com o valor do primeiro elemento. $T(n) = 3\lfloor n \rfloor/2$.
- Se n for par: iniciar *min* com o menor valor do primeiro par e *max* com o maior valor do primeiro par $T(n) = 3\lfloor n \rfloor/2 - 2$.

Ambos algoritmos são $\Theta(n)$, no entanto esta versão executa em tempo menor, logo ele é mais eficiente.

Não podemos nos confundir, ambos são ótimos

Problema MinMax

- Processe os elementos aos pares, para cada par, compare o menor com o valor mínimo atual e o maior com o valor máximo atual. São 3 comparações para cada 2 elementos.
- Se n for ímpar: iniciar os valores *min* e *max* com o valor do primeiro elemento. $T(n) = 3\lfloor n \rfloor/2$.
- Se n for par: iniciar *min* com o menor valor do primeiro par e *max* com o maior valor do primeiro par $T(n) = 3\lfloor n \rfloor/2 - 2$.

Ambos algoritmos são $\Theta(n)$, no entanto esta versão executa em tempo menor, logo ele é mais eficiente.

Não podemos nos confundir, ambos são ótimos

Atividades baseadas no CLRS.

1. Resolver o problema 2-2.
2. Faça um algoritmo melhor para o *Bubblesort* de forma que o tempo de execução no melhor caso seja $\Theta(n)$
3. Qual o tempo de execução de seu algoritmo no caso médio?