“A matemática requer uma dose pequena, não de genialidade, mas de liberdade de imaginação, que em uma dose maior, seria loucura” (Angus K. Rodgers).

Eficiência, Invariantes e Recorrências

Paulo Ricardo Lisboa de Almeida

Alguns conteúdos e exemplos foram baseados nas aulas do Professor Renato Carmo - www.inf.ufpr.br/renato

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Eficiência

Na Ciência da Computação buscamos algoritmos **eficientes** para resolver os problemas.

Vamos estabelecer eficiência como: recursos consumidos/quantidade de tarefa realizada

Mas que recursos são esses que desejamos gastar eficientemente?

Exemplos de Recursos

Joules

Tempo

Memória

Transferência de dados (comunicação)

...

Eficiência

Eficiência de algoritmos e análise de recursão via recorrência são temas estudados profundamente em **Matemática Discreta** e em **Análise de Algoritmos**

Mas vamos olhar para esses temas superficialmente para entender melhor nossos algoritmos

Muitas das provas apresentadas serão **informais**

Servirão como uma introdução ao tema

Valor Mínimo

Para encontrar o valor mínimo de um vetor, precisamos realizar comparações

A questão é, **quantas comparações entre elementos são necessárias?**

Vamos definir a quantidade de comparações como o recurso sendo medido.

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Obs.: essa é a versão **iterativa**

Valor Mínimo

Para um vetor *v* de tamanho $|v| = n = b-a+1$

O loop inicia as comparações em $a+1$, e termina em b

São realizadas então $b - (a + 1) +1 = b-a$ comparações entre elementos

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Valor Mínimo

Para um vetor v de tamanho $|v| = n = b-a+1$

O loop inicia as comparações em $a+1$, e termina em b

São realizadas então $\underbrace{b}_{\text{Termina em } b} - \underbrace{(a+1)}_{\text{Inicia em } a+1} + 1 = b-a$ comparações entre elementos

Outra forma é pensar que o loop começa em a+1, e termina em b. Assim temos *b-a* comparações.

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
        m ← j
retorne m
```

Valor Mínimo

Para um vetor *v* de tamanho $|v| = n = b-a+1$

O loop inicia as comparações em $a+1$, e termina em b

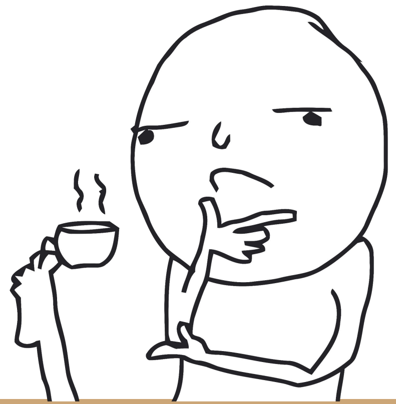
São realizadas então $b - (a + 1) +1 = b-a$ comparações entre elementos

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Invariantes

Como sabemos que esse algoritmo realmente funciona?

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```



Invariantes

Para começar, vamos fazer um teste de mesa com

v = [28, 12, 1, 25, 10, 1]

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Invariantes

Note o loop j < b

1. Toda vez que passamos pelo loop, já foram avaliados os índices de *a* a *j* de *v*, ou seja, analisamos a instância do problema *(v,a,j)*

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Invariantes

Note o loop j < b

1. Toda vez que passamos pelo loop, já foram avaliados os índices de *a* a *j* de *v*, ou seja, analisamos a instância do problema *(v,a,j)*
2. Então *m* é uma resposta da instância menor do problema, *(v,a,j)*

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Invariantes

Note o loop j < b

1. Toda vez que passamos pelo loop, já foram avaliados os índices de *a* a *j* de *v*, ou seja, analisamos a instância do problema *(v,a,j)*
2. Então *m* é uma resposta da instância menor do problema, *(v,a,j)*
3. A propriedade é mantida para toda nova iteração do loop

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Invariantes

Note o loop j < b

1. Toda vez que passamos pelo loop, já foram avaliados os índices de *a* a *j* de *v*, ou seja, analisamos a instância do problema *(v,a,j)*
2. Então *m* é uma resposta da instância menor do problema, *(v,a,j)*
3. A propriedade é mantida para toda nova iteração do loop
4. Quando o loop termina *j = b* e então m é uma resposta da instância (v, a, b) do problema

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Invariantes

Provar que um algoritmo está correto via **invariantes** envolve três passos

Demonstrar que a **inicialização** está correta

Demonstrar que a **manutenção** está correta (a cada iteração, as coisas permanecem corretas)

Demonstrar que a **terminação** está correta

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Invariantes

Inicialização: no início j=a, e o vetor (v,a,a) tem apenas um elemento. v = a que é o índice do único elemento do vetor.

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Invariantes

Inicialização: no início j=a, e o vetor (v,a,a) tem apenas um elemento. v = a que é o índice do único elemento do vetor.

manutenção: a cada iteração, um novo elemento é adicionado no vetor, que está na forma (v, a, j). Antes da iteração, m é o menor elemento do vetor (v,a,j-1). O novo elemento v[j] é comparado com v[m]. Se v[j] < v[m], m é atualizado para j, caso contrário, o novo elemento adicionado no vetor não é menor que qualquer elemento do vetor (v,a,j-1).

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Invariantes

Inicialização: no início j=a, e o vetor (v,a,a) tem apenas um elemento. v = a que é o índice do único elemento do vetor.

manutenção: a cada iteração, um novo elemento é adicionado no vetor, que está na forma (v, a, j). Antes da iteração, m é o menor elemento do vetor (v,a,j-1). O novo elemento v[j] é comparado com v[m]. Se v[j] < v[m], m é atualizado para j, caso contrário, o novo elemento adicionado no vetor não é menor que qualquer elemento do vetor (v,a,j-1).

Terminação: quando j = b, o algoritmo termina depois de analisar o vetor (v, a, b)

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
j ← a
m ← a
enquanto j < b
    j ← j + 1
    se v[j] < v[m]
       m ← j
retorne m
```

Recorrências

Analisando a versão **recursiva** da função minimoVetor

Vamos responder quantas comparações são necessárias nessa versão do algoritmo

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
se a = b
    retorne a
m ← minimoVetor(v,a,b-1)
Se v[b] < v[m]
    m ← b
retorne m
```

Recorrências

Primeiro, lembrando que o tamanho do vetor é $|(v, a, b)| = b - a + 1 = n$

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
se a = b
    retorne a
m ← minimoVetor(v,a,b-1)
Se v[b] < v[m]
     m ← b
retorne m
```

Recorrências

Primeiro, lembrando que o tamanho do vetor é $|(v, a, b)| = b - a + 1 = n$

Para o caso base, qual o valor de *n*, e quantas **comparações de elementos do vetor** são necessárias?

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
se a = b
    retorne a
m ← minimoVetor(v,a,b-1)
Se v[b] < v[m]
     m ← b
retorne m
```

Recorrências

Primeiro, lembrando que o tamanho do vetor é $|(v,a,b)| = b - a + 1 = n$

Para o caso base, qual o valor de *n*, e quantas **comparações de elementos do vetor** são necessárias?

n = 1

São necessárias 0 comparações entre elementos

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
se a = b
    retorne a
m ← minimoVetor(v,a,b-1)
Se v[b] < v[m]
     m ← b
retorne m
```

Recorrências

E para os casos não base?

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
se a = b
    retorne a
m ← minimoVetor(v,a,b-1)
Se v[b] < v[m]
     m ← b
retorne m
```

Recorrências

E para os casos não base?

$n > 1$

São necessárias 1 + numeroComparacoes(minimoVetor(v,a,b-1))

```
função minimoVetor (v,a,b)
entrada: vetor v indexado por [a..b], com a ≤ b
saída: m ∈ [a..b], tal que v[m] ≤ v[j] ∀j ∈ [a..b]
se a = b
    retorne a
m ← minimoVetor(v,a,b-1)
Se v[b] < v[m]
     m ← b
retorne m
```

Recorrências

Definindo uma função de custo $C(n)$

Recorrências

Definindo uma função de custo $C(n)$

$$C(n) = \begin{cases} 0, & \text{se } n = 1, \\ 1 + C(n-1), & \text{se } n > 1, \end{cases}$$

Recorrências

C(n) é uma função definida de forma recursiva. Para calcular C(n) precisamos calcular C(n-1), mas para calcular C(n-1) precisamos calcular C(n-2), ...

Isso é uma **recorrência**.

$$C(n) = \begin{cases} 0, & \text{se } n = 1, \\ 1 + C(n-1), & \text{se } n > 1, \end{cases}$$

Recorrências

Exemplo: qual o custo para C(5)?

$$C(n) = \begin{cases} 0, & \text{se } n = 1, \\ 1 + C(n-1), & \text{se } n > 1, \end{cases}$$

Recorrências

Vamos expandir a função para resolver a equação para o caso geral C(n)

$$\forall n > 0,$$

$$C(n) = 1 + C(n-1) = 1 + (1 + C(n-2)) = 1 + (1 + (1 + C(n-3))) = \dots$$

$$= C(n) = 1 + C(n-1) = 2 + C(n-2)) = 3 + C(n-3))) = \dots$$

$$= \mu + C(n-\mu)$$

$$C(n) = \begin{cases} 0, & \text{se } n = 1, \\ 1 + C(n-1), & \text{se } n > 1, \end{cases}$$

Recorrências

Continuamos recursivamente, até atingir o caso base, onde sabemos a resposta

Temos o caso base para n = 1, logo

$$n - \mu = 1$$

Implicando assim que

$$\mu = n - 1$$

$$C(n) = \begin{cases} 0, & \text{se } n = 1, \\ 1 + C(n-1), & \text{se } n > 1, \end{cases}$$

Substituindo

$$C(n) = \mu + C(n - \mu)$$

$$\mu = n - 1$$

$$C(n) = \begin{cases} 0, & \text{se } n = 1, \\ 1 + C(n-1), & \text{se } n > 1, \end{cases}$$

$$C(n) = \mu + C(n - \mu) = n - 1 + C(n - (n - 1))$$

$$= n - 1 + C(1) = n - 1$$

Comparando

Comprovamos então que tanto a versão iterativa quanto a versão recursiva do algoritmo para encontrar o índice do menor valor custam *n-1* comparações.

Mais

Você vai ver recorrências em mais detalhes em outras disciplinas, como Matemática Discreta e Análise de Algoritmos

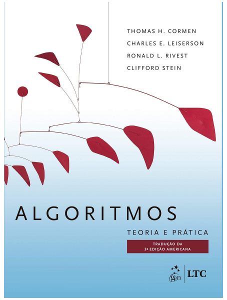
A prova da corretude de algoritmos recursivos geralmente envolve o conceito de indução, que também é tema dessas disciplinas.

Exercícios

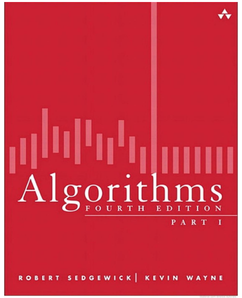
1. Considere o algoritmo para calcular n! (dado na aula passada).
   a. Na versão **iterativa** do algoritmo, encontre a invariante e demonstre através dela que o algoritmo está correto
   b. Na versão **recursiva** do algoritmo, expresse M(n) como uma recorrência para calcular o número de multiplicações realizadas.
   c. Resolva a recorrência
2. Considere o algoritmo para calcular o somatório dos elementos de um vetor (exercício da aula passada).
   a. Na versão **iterativa** do algoritmo, encontre a invariante e demonstre através dela que o algoritmo está correto
   b. Na versão **recursiva** do algoritmo, expresse S(n) como uma recorrência para calcular o número de somas realizadas.
   c. Resolva a recorrência

Referências

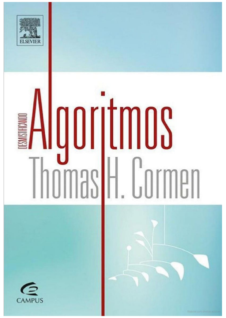
T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos: Teoria e Prática. 3a ed. 2012



R. Sedgewick, K. Wayne. Algorithms Part I. 4a ed. 2014



T. Cormen. Desmistificando algoritmos. 2017.



H. Schildt. C completo e total. 1996

Licença