Projeto e Análise de Algoritmos

Correção de algoritmos

Lehilton Pedrosa

Primeiro Semestre de 2020

Invariante de laço

Invariante de laço

- Correção de INSERTION-SORT

Revendo o Insertion-Sort

```
INSERTION-SORT(A, n)
1   para j ← 2 até n faça
2       chave ← A[j]
3       i ← j − 1
4       enquanto i ≥ 1 e A[i] > chave faça
5           A[i + 1] ← A[i]
6           i ← i − 1
7       A[i + 1] ← chave
```

Até agora:

- já vimos que o algoritmo termina
- e analisamos sua complexidade de tempo

O que falta fazer?

- Verificar se ele produz uma resposta correta.

Invariante de laço

Definição

Uma invariante laço é uma propriedade que

- depende dos valores das variáveis
- está associada a determinada posição de um laço
- é satisfeita em toda execução do laço

A posição escolhida é normalmente descrita como

- imediatamente antes ou depois da iteração do laço
- imediatamente antes ou depois de determinada linha

Objetivos

- após o término do laço, deve ser uma propriedade útil para se mostrar a correção do algoritmo
- permite nos concentrar apenas em uma iteração do laço

Exemplo de invariante

```
INSERTION-SORT(A, n)
1   para j ← 2 até n faça
2       chave ← A[j]
3       i ← j − 1
4       enquanto i ≥ 1 e A[i] > chave faça
5           A[i + 1] ← A[i]
6           i ← i − 1
7       A[i + 1] ← chave
```

Invariante 1

Imediatamente antes de cada iteração do laço **para**,
o subvetor $A[1 \ldots j-1]$ está ordenado.

- posição da invariante: antes da iteração do laço **para**
- propriedade invariante: $A[1 \ldots j-1]$ está ordenado

Demonstrando uma invariante

Tipicamente, demonstramos uma invariante com as etapas:

1. mostre que a propriedade vale antes de qualquer iteração
2. mostre que, se a propriedade vale no início da iteração, então ela também vale no final da iteração
3. conclua que a invariante vale quando o laço termina

Estamos usando o princípio da indução!

- a base corresponde à etapa 1
- o passo indutivo corresponde à etapa 2

Demonstrando uma invariante: caso base

Considere a primeira iteração do laço **para**

- no início da iteração, $j = 2$
- assim, o subvetor $A[1 \ldots j-1]$ contém apenas um elemento
- então, a invariante vale antes de qualquer iteração

Demonstrando uma invariante: passo indutivo

Suponha que a invariante vale no início de alguma iteração

- nessa iteração, temos $chave = A[j]$
- após o laço **enquanto**, inserimos *chave* na posição $i + 1$
- quando inserirmos a *chave*, queremos
  1. que os anteriores sejam menores e estejam ordenados
  2. que os posteriores sejam maiores e estejam ordenados
- vamos criar uma sub-invariante para o laço **enquanto**!

Sub-invariante

Sub-invariante

Imediatamente antes de cada iteração do laço **enquanto**:

1. $A[1 \ldots j]$ está ordenado
2. $chave \leq A[i+1]$

Demonstração:

- Antes de qualquer iteração, as afirmações valem
- Suponha que valem no início de uma iteração
  - pela condição do laço, $chave < A[i]$ e $A[i]$ não se altera
  - como diminuímos o valor de $A[i+1]$ para $A[i]$, o vetor $A[1 \ldots j]$ continua ordenado
- Assim, as afirmações mantêm-se no final

Demonstrando uma invariante: passo indutivo (cont)

Quando o laço **enquanto** termina

- pela sub-invariante, o vetor $A[1 \ldots j]$ está ordenado
- também pela sub-invariante, $chave \leq A[i+1]$
- se paramos porque $i = 0$, $chave \leq A[1]$
- do contrário, paramos porque $A[i] < chave$
- em qualquer caso, mantemos $A[1 \ldots j]$ ordenado
- então, a invariante vale quando o laço **para** termina

Outra invariante

- já demonstramos a Invariante 1
- o laço termina apenas quando $j = n + 1$
- pela invariante, nesse momento $A[1 \ldots n]$ está ordenado
- mais isso não é suficiente, pois o vetor poderia ser

20	20	20	20	20	20	20	20	20	20	20

Invariante 2

Imediatamente antes de cada iteração do laço **para**,
o subvetor $A[1 \ldots n]$ é uma permutação dos dados da entrada.

Exercício: demonstre essa invariante.

Demonstrando a correção do algoritmo

Suponha que já demonstramos as Invariantes 1 e 2

Teorema

O algoritmo INSERTION-SORT está correto.

Demonstração:

- Quando o laço termina, temos $j = n + 1$, então a Invariante 1 implica que $A[1 \ldots n]$ está ordenado
- Nesse momento, a Invariante 2 implica que o vetor $A[1 \ldots n]$ contém todos elementos da entrada.
- Portanto, o vetor devolvido é uma ordenação do vetor de entrada.

Invariante de laço

- Conversão binária

Conversão para representação binária

Problema: conversão binária
Entrada:

- um número inteiro não negativo n

Saída:

- vetor B com representação binária (invertida) de n

```
CONVERTE-BINÁRIO(n)
1  t ← n
2  k ← −1
3  enquanto t > 0 faça
4      k ← k + 1
5      B[k] ← t mod 2
6      t ← t div 2
7  retorne B.
```

Invariante para Converte-Binário

Invariante

No início da iteração do laço **enquanto**:

$$n = t \cdot 2^{k+1} + \sum_{i=0}^{k} 2^i \cdot B[i].$$

Demonstração:

- Antes de qualquer iteração, temos $k = -1$, $t = n$
- Assim,

$$n = t \cdot 1 + 0 = t \cdot 2^{k+1} + \sum_{i=0}^{k} 2^i \cdot B[i]$$

- Portanto, a afirmação vale no início do laço

Invariante para Converte-Binário (cont)

Suponha que a invariante vale no início da iteração

- Sejam $k' = k + 1$ e $t' = t \operatorname{div} 2$
- Sabemos que $B[k+1] = t \bmod 2$
- Como a afirmação vale no início,

$$\begin{aligned} n &= t \cdot 2^{k+1} + \sum_{i=0}^{k} 2^i \cdot B[i] \\ &= (t \operatorname{div} 2) \cdot 2^{k+2} + 2^{k+1} \cdot (t \bmod 2) + \sum_{i=0}^{k} 2^i \cdot B[i] \\ &= t' \cdot 2^{k+2} + 2^{k+1} \cdot B[k+1] + \sum_{i=0}^{k} 2^i \cdot B[i] \\ &= t' \cdot 2^{(k+1)+1} + \sum_{i=0}^{k+1} 2^i \cdot B[i] \\ &= t' \cdot 2^{k'+1} + \sum_{i=0}^{k'} 2^i \cdot B[i] \end{aligned}$$

- Portanto, a invariante vale no final da iteração

Correção de algoritmos recursivos

Correção de algoritmos recursivos

- Algoritmo MERGE-SORT

Correção de MERGE-SORT

```
MERGE-SORT(A, p, r)
1   se p < r
2       então q ← ⌊(p + r)/2⌋
3             MERGE-SORT(A, p, q)
4             MERGE-SORT(A, q + 1, r)
5             INTERCALA(A, p, q, r)
```

Como demonstrar a correção de um algoritmo recursivo?

- podemos usar indução diretamente
- precisamos verificar as demais sub-rotinas

Correção de MERGE-SORT

- queremos mostrar que MERGE-SORT ordena $A[p \ldots r]$
- basta usar indução em $n = r - p + 1$

Revendo Intercala

```
INTERCALA(A, p, q, r)
 1  para i ← p até q faça
 2      B[i] ← A[i]
 3  para j ← q + 1 até r faça
 4      B[r + q + 1 − j] ← A[j]
 5  i ← p
 6  j ← r
 7  para k ← p até r faça
 8      se B[i] ≤ B[j]
 9          então A[k] ← B[i]
10                i ← i + 1
11          senão A[k] ← B[j]
12                j ← j − 1
```

Invariante

Em cada iteração da linha 7, vale:

1. $A[p \ldots k-1]$ está ordenado,
2. $A[p \ldots k-1]$ contém itens de $B[p \ldots i-1]$ e de $B[j+1 \ldots r]$,
3. $B[i] \geq A[k-1]$ e $B[j] \geq A[k-1]$.

Exercício: demonstre

- a invariante
- a correção de Intercala

Correção de Merge-Sort

Base da indução:

- para um vetor de tamanho 0 ou 1, o vetor não é alterado
- assim Merge-Sort está correto nesses casos

Caso geral:

- considere um vetor de tamanho n
- suponha que Merge-Sort ordena vetores menores
- daí, após as chamadas recursivas, os subvetores $A[p \ldots q]$ e $A[q+1 \ldots r]$ estão ordenados
- como Intercala está correto, o vetor $A[p \ldots r]$ estará ordenado no final do algoritmo