CORRETUDE DE ALGORITMOS

Profª. Thaís Alves Burity Rocha

Agenda

- Invariantes de laço
- Corretude de algoritmos não-recursivos (iterativos)
  - Soma
  - Fatorial
  - BubbleSort
  - Potência
  - Máximo valor

Invariante de laço

- Nos auxilia a analisar algoritmos, atestando sua corretude

- Pode ser definido como uma relação entre as variáveis de um algoritmo que é verdadeira durante as seguintes condições:
  - Inicialização: Antes do início do laço (loop)
  - Manutenção: Durante a execução do loop
  - Término: Na saída do loop

Exemplo 1: Soma

```
Soma(A[])
s ← 0
i ← 1
while i ≤ A.length do
        s ← s + A[i]
        i ← i + 1
return s
```

Invariante: $s = \sum_{k=1}^{i-1} A[k]$

Exemplo 1: Corretude

- Inicialização:
  - Antes da inicialização **i = 1**, assim:
  - **s = 0**, uma vez que a soma dos elementos de um vetor sem nenhum elemento é 0
- Manutenção:
  - Suponha que o invariante está correto, ou seja, **s = A[1] + A[2] + ...+ A[i-1]**. Então, durante o loop, o elemento **A[i]** é somado a **s** e, logo após, o valor **i** é incrementado de **1**. Portanto, isto mostra que, depois de uma iteração, o invariante se mantém.

Exemplo 1: Corretude

- Término:
  - O loop só termina quando **i>A.length**, isto é, quando **i=A.length + 1**. Substituindo **i** por **A.length +1** no invariante, temos que **A[1] + A[2] + ... + A[A.length]** são os elementos já somados, isto é, o vetor inteiro, o que mostra que o algoritmo está correto.

Ainda sobre laços

- Dois elementos:
  - Guarda: expressão booleana que indica se o corpo do laço deve ou não ser executado
  - Variante: expressão numérica que indica o número de iterações que ainda resta ocorrer

```
Soma(A[])
s ← 0
i ← 1
while i ≤ A.length do
        s ← s + A[i]
        i ← i + 1
return s
```

Guarda: $i \leq A.length$
Variante: $A.length - i + 1$

Ainda sobre laços

- Um laço está correto se as seguintes 5 condições são satisfeitas:
  - Início: invariante é verdadeiro antes de entrar no laço
  - Invariância: o corpo do laço preserva o invariante
  - Progresso: o corpo do laço diminui a variante
  - Limitação: quando a variante atinge certo valor, a guarda se torna falsa
  - Saída: a negação da guarda e o invariante descrevem o objetivo do laço

Exemplo 2: Fatorial

```
Fatorial (n)
    fat = 1
    i = 1
    while (i≤n) {
      fat = fat × i
      i = i + 1
    }
    return fat
```

- Qual a invariante?
- Prove que o algoritmo está correto.

Exemplo 2: Fatorial

```
Fatorial (n)
    fat = 1
    i = 1
    while (i≤n) {
      fat = fat × i
      i = i + 1
    }
    return fat
```

Invariante: $\text{fat} = \prod_{k=1}^{i-1} k$

Exemplo 2: Fatorial

- Início:
  - Antes da inicialização **i = 1**, assim:
  - **fat = 1**. Mostrando que o invariante está correto antes da inicialização.
- Invariância:
  - Suponha que o invariante está correto, então, durante o loop, o algoritmo multiplica o fatorial de **i-1** por **i**, e logo após incrementa **i** de **1**. Portanto, isto mostra que, depois de uma iteração, o invariante se mantém.

Exemplo 2: Fatorial

- Progresso:
  - A variante do loop é **n - i +1**. Como **i** é sempre incrementado no final do loop, a variante é decrementada a cada iteração. Isto mostra que o algoritmo termina.
- Término (e Limitação):
  - O loop termina quando **i > n**, isto é, **i = n+1**. Substituindo **i** por **n+1** no invariante, temos que:

$$fat = \prod_{k=1}^{n} k = 1 \times 2 \times \ldots \times n = n!$$

Mostrando que o algoritmo está correto.

Laços aninhados

- Analisar um laço por vez, começando pelo mais interno
- Para cada laço, determinar um **invariante de laço (IL)**
- Provar que o invariante de laço é válido
- Mostrar que o algoritmo termina
- Usar o invariante de laço para provar que o algoritmo retorna o valor desejado

Bubblesort

```
bubblesort(A[], n){
    for (x=1; x≤n; x++){                  Loop externo (2)
        for (y=n; y≥x+1; y--){            Loop interno (1)
            if (A[y] < A[y-1])
                swap(A[y],A[y-1])
        }
    }
}
```

- Enuncie uma propriedade invariante mantida pelo laço “for” interno e demonstre sua validade.
- Usando a condição de término garantida pela propriedade invariante que você enunciou e demonstrou no item anterior para o laço “for” interno, enuncie e demonstre uma propriedade invariante do laço “for” externo que garanta que, ao final do algoritmo, os elementos do vetor **A** estarão dispostos em ordem crescente.

Corretude do Bubblesort

- Loop interno
  - Invariante:
    - O elemento **A[y]** é o menor elemento do sub-vetor **A[y...n]**
  - Início:
    - Antes da inicialização **y = n**, assim:
    - Substituindo este valor no invariante temos o sub-vetor **A[n...n]** que possui apenas **1** elemento. Assim, obviamente podemos considerar que **A[n]** é o menor elemento.

Corretude do Bubblesort

- Loop interno
  - Manutenção/Invariância:
    - Suponha que o invariante está correto, então **A[y]** é o menor elemento do sub-vetor **A[y...n]**. O loop interno permuta o conteúdo de **A[y-1]** e **A[y]**, se **A[y-1]** for maior que **A[y]** e, logo após, decrementa **y** de **1**, mostrando que o invariante se mantém.
  - Término (e Limitação):
    - O loop termina quando **y=x**. Substituindo **y** por **x** no invariante, temos que **A[x]** é o menor elemento do sub-vetor **A[x...n]**.

Corretude do Bubblesort

- Loop externo
  - Invariante:
    - O sub-vetor **A[1 ... x-1]** está ordenado
  - Início:
    - Antes da inicialização **x = 1**, assim:
    - O sub-vetor não possui nenhum elemento, assim podemos considerar que ele está ordenado.

Corretude do Bubblesort

- Loop externo
  - Manutenção:
    - Suponha que o invariante está correto, então os elementos **A[1... x-1]** já estão ordenados. Como foi mostrado, o loop interno permuta os elementos do sub-vetor **A[x ... n]** até que o elemento **A[x]** seja o menor elemento deste sub-vetor, o que mantém o segundo invariante, uma vez que **x** é incrementado de **1** logo após isso.
  - Término:
    - O loop termina quando **x=n+1**. Substituindo **x** por **n+1** no invariante, temos que o sub-vetor **A[1...n]**, mostrando que ao final do algoritmo todo vetor está ordenado.

Exercício 1: Potência

- Fazer algoritmo para calcular iterativamente potência de um número: x^n

- Prove que o algoritmo está correto usando invariantes

Exercício 1: Potência (Solução)

- Fazer algoritmo para calcular iterativamente potência de um número: x^n

```
Potencia(x, n)
p ← 1
i ← 0
while i < n do
        p ← p.x
        i ← i + 1
return p
```

Exercício 1: Potência (Solução)

- Invariante: **$p = x^i$**
- Início:
  - Antes da inicialização **$i = 0$**, assim:
  - **$p = 1$**. Mostrando que o invariante está correto antes da inicialização, já que **$x \cdot x^{-1} = 1$** e de fato, **$x^0 = 1$**
- Invariância/Manutenção:
  - Suponha que o invariante está correto, então, durante o loop, o algoritmo multiplica **p** por **x^i**, e logo após incrementa **i** de **1**. Portanto, isto mostra que, depois de uma iteração, o invariante se mantém.

Exercício 1: Potência (Solução)

- Progresso:
  - A variante do loop é **n - i**. Como **i** é sempre incrementado no final do loop, a variante é decrementada a cada iteração. Isto mostra que o algoritmo termina.
- Término (e Limitação):
  - O loop só termina quando **$i \geq n$**, isto é, quando **$i=n$**. Substituindo **i** por **n** no invariante, temos que **$p=x^n$**, mostrando que o algoritmo está correto.

Exercício 2: Máximo valor

- Prove que o algoritmo está correto usando invariantes

```
MAXIMO(A, n)
max = A[1]
for i = 2 to n
        if A[i] > max
                max = A[i]
return max
```

Exercício 2: Máximo valor (Solução)

- Invariante: **max** contém o maior valor do sub-vetor **A[1… i-1]**
- Início:
  - Antes da inicialização **i = 2**
  - Substituindo este valor no invariante, temos o sub-vetor **A[1...1]** que possui apenas **1** elemento. Assim, obviamente podemos considerar que **A[1]** é o maior valor.
- Invariância/Manutenção:
  - Suponha que o invariante está correto, então, durante o loop, dado um valor de **i**, o valor de **A[i]** é atribuído à **max** se **A[i]>max** e, logo após, **i** é incrementado de **1**. Logo, para o sub-vetor **A[1...i-1]**, sabe-se que **max** possui o maior valor.

Exercício 2: Máximo valor (Solução)

- Progresso:
  - A variante do loop é **n - i +1**. Como **i** é sempre incrementado no final do loop, a variante é decrementada a cada iteração. Isto mostra que o algoritmo termina.
- Término (e Limitação):
  - O loop só termina quando **i>n**, isto é, quando **i=n+1**. Substituindo **i** por **n+1** no invariante, temos que **A[1...n+1-1] = A[1...n]**. Logo, **max** contém o maior valor no vetor A[1...n].

Referência

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: Teoria e prática. Tradução da Segunda edição Americana. Editora Campus, 2002.