# Análise de Algoritmos – Tópico 10

Prof. Dr. Juliano Henrique Foleis

## Prova de Correção por Invariante de Laço

Umas das técnicas de prova de correção de algoritmos consiste na formulação das invariantes de laço, que servem para mostrar que determinada propriedade, denominada invariante de laço, é verdadeira após a execução de um laço de repetição. Para que a invariante de laço sirva para mostrar que o algoritmo está correto, ela deve ser enunciada de forma que a propriedade seja relevante na relação entrada-saída do problema resolvido pelo algoritmo. Para que a prova seja válida, a invariante deve ser verdadeira em três momentos da execução do laço:

- **Inicialização** – A invariante deve ser verdadeira antes da primeira iteração de um laço, logo após os procedimentos de inicialização;
- **Manutenção** – A invariante é verdadeira antes de uma iteração qualquer do laço, e mantém-se verdadeira antes da próxima iteração;
- **Término** – Quando a execução de um laço termina, a invariante nos dá uma propriedade útil que ajuda a provar que o algoritmo está correto.

Quando as duas primeiras propriedades se mantém, a invariante de laço é verdadeira antes de cada iteração do laço. Note que outros fatos provados também podem ser usados na prova, inclusive para demonstrar que a invariante de laço se mantém verdadeira antes da próxima iteração (de forma análoga ao passo de indução em uma prova por indução).

Há certa similaridade entre a indução matemática e o uso de invariantes de laço. Para provar que uma propriedade se mantém, um caso base é provado, além do passo da indução. Neste caso, ao mostrar que a invariante é verdadeira antes do laço corresponde ao caso base, enquanto mostrar que a invariante se mantém de iteração a iteração corresponde ao passo indutivo. A terceira propriedade é provavelmente a mais importante, uma vez que a invariante de laço está sendo utilizada para provar que o algoritmo está correto e que o algoritmo pára quando a relação entrada-saída é obtida. Normalmente a invariante é utilizada com a condição de término do laço. A propriedade de término é diferente em relação ao uso de indução, no qual o passo indutivo é aplicado infinitamente. No caso da prova por invariante de laço, a "indução" termina quando o laço termina.

Em resumo, a invariante de laço possibilita testar a correção de um algoritmo e pode ser definida como uma relação entre variáveis de um algoritmo que é verdadeira durante a inicialização, execução (manutenção) e término de laços de repetição.

**Exemplo 1**- Correção da Ordenação Por Inserção (*InsertionSort*)

Este algoritmo possui laços aninhados. Neste caso, devemos provar que os laços estão corretos, de "dentro para fora". Vamos considerar que o laço interno (WHILE) está correto, e que, ao final de cada execução dele, os elementos $A[1 \dots i] \leq \text{KEY}$ e que $A[i+1 \dots n] > \text{KEY}$.

```
INSERTIONSORT(A)
1. FOR j = 2 TO A.size DO
2.    KEY = A[j]
3.    i = j - 1
4.    WHILE i > 0 AND A[i] > KEY DO
5.       A[i+1] = A[i]
6.       i = i - 1
7.    END WHILE
8.    A[i+1] = KEY
9. END FOR
```

No início de cada iteração do laço `for`, $j$ é o elemento que está sendo inserido no subvetor $A[1 \dots j-1]$, que já está ordenado. O restante do vetor, $A[j+1 \dots n]$, corresponde aos elementos não ordenados. Na realidade, os elementos $A[1 \dots j-1]$ são os elementos originalmente nas posições 1 até $j-1$, mas em ordem crescente. Destes fatos

é possível enunciar uma invariante de laço que pode ser utilizado para provar a correção da ordenação por inserção.

**Invariante de Laço:** No início de cada iteração do laço FOR das linhas $1-9$, o subvetor $A[1 \ldots j-1]$ consiste nos elementos originais de $A[1 \ldots j-1]$, mas em ordem crescente.

Para mostrar que o laço produz os resultados esperados, devemos mostrar que a invariante de laço é válida durante a inicialização, manutenção e término da execução do laço.

**Inicialização** – Iniciamos mostrando que a invariante de laço é verdadeira antes da primeira iteração, quando $j = 2$. Desta forma, o subvetor $A[1 \ldots j-1]$ consiste em apenas 1 elemento, $A[1]$, que é o elemento original na posição 1. Além disto, este subvetor está ordenado, pois um subvetor unitário sempre está ordenado. Isto mostra que a invariante de laço é verdadeira antes da primeira iteração do laço.

**Manutenção** – Devemos mostrar que a propriedade da invariante de laço é mantida verdadeira antes do início de cada iteração. Para isto, mostremos que as modificações realizadas durante determinada iteração mantém a invariante de laço verdadeira para o início da próxima iteração.

Pela invariante de laço, no início da iteração o subvetor $A[1 \ldots j-1]$ consiste nos elementos originais de $A[1 \ldots j-1]$, mas em ordem crescente. O elemento da posição $j$ pode ou não estar na posição adequada. Consideramos que o laço "WHILE" está funcionando corretamente. Informalmente, o corpo do laço "FOR" funciona copiando $A[j-1], A[j-2], \ldots$ uma posição para frente até que encontre a posição adequada para $A[j]$. Desta forma, após a execução da iteração, o elemento inicialmente na posição $j$ estará inserido em sua posição final ($i+1$, encontrada pelo "WHILE"), fazendo com que agora o subvetor $A[1 \ldots j]$ tenha em elementos originalmente em $A[1 \ldots j]$, mas em ordem crescente. Ao incrementar $j$, a propriedade da invariante de laço é reestabelecida.

**Término** – A condição que provoca o término do laço é $j > A.size$. Como a cada iteração é acrescido de 1, temos no final que j = n+1 no término do laço. Substituindo j por n+1 na invariante, temos que o subvetor A[1..n] consiste nos elementos originalmente contidos em A[1..n], mas em sequência ordenada. Como esse subvetor corresponde ao vetor inteiro, conclui-se que o arranjo está ordenado.

Logo, o algoritmo está correto.

# Exemplos

Exemplos de correção por invariante de laço dos algoritmos de soma elementos do vetor, fatorial, potência e valor máximo de vetor.

## Exemplo 1: Soma elementos de vetor

```
1: function SomaElementos(V, n)
2:     soma=0
3:     i=1
4:     while i ≤ n do
5:         soma = soma + V[i]
6:         i = i + 1
7:     end while
8:     return soma
9: end function
```

**Invariante:**

$$soma = \sum_{k=1}^{i-1} V[k]$$

ou de forma textual:

*Antes do início de cada iteração do* `while`, $soma = V[1] + V[2] + ... + V[i-1]$.

**Prova de correção por invariante de laço:**

**Inicialização:** Antes da primeira iteração $i = 1$, logo $i-1$ implica em um vetor vazio, e a soma dos elementos de um vetor vazio é 0. Assim, a invariante é verdadeira na inicialização, pois $soma = 0$ antes da execução da

primeira linha interna do laço.

**Manutenção:** Supondo que a invariante está correta, ou seja, $soma = V[1] + V[2] + ... + V[i-1]$, na linha 5 o elemento $V[i]$ é somado a *soma* e, em seguida, i é incrementado em 1 (linha 6). Verificando a invariante temos que $soma = V[1] + V[2] + ... + V[i-1] + V[i]$, mas como i é incrementado, $soma = V[1] + V[2] + ... + V[i-2] + V[i-1]$, o que mostra que depois de uma iteração, o invariante se mantém.

**Término:** O laço é finalizado quando $i > n$, ou seja, $i = n + 1$. Substituindo $i$ por $n + 1$ na invariante, temos que: $soma = V[1] + V[2] + ... + V[n + 1 - 1]$, ou seja, $soma = V[1] + V[2] + ... + V[n]$, que corresponde a soma de todos os elementos do vetor. Logo, o algoritmo está correto.

## Exemplo 2: Fatorial

```
1: function FATORIAL(n)
2:     fatorial=1
3:     i=1
4:     while i ≤ n do
5:         fatorial = fatorial * i
6:         i = i + 1
7:     end while
8:     return fatorial
9: end function
```

**Invariante:**

$$fatorial = \prod_{k=1}^{i-1} k$$

ou de forma textual:

*Antes do início de cada interação do* `while`, $fatorial = 1 \times 2 \times ... \times i - 1$.

**Prova de correção por invariante de laço:**

**Inicialização:** Antes da primeira iteração, $i = 1$ e $fatorial = 1$, logo $i - 1$ implica em 0! que é 1, pois a invariante pode ser escrita como $fatorial = (i - 1)!$. Assim, a invariante é verdadeira na inicialização.

**Manutenção:** Supondo que a invariante está correta, ou seja, $fatorial = 1 \times 2 \times ... \times i - 1$, $fatorial$ recebe o valor da multiplicação de $i$ por $fatorial$ (linha 5), em seguida, $i$ é incrementado em 1 (linha 6). Verificando a invariante temos que $fatorial = 1 \times 2 \times ... \times i - 1 \times i$, mas como i é incrementado, $fatorial = 1 \times 2 \times ... \times i - 2 \times i - 1$, o que mostra que depois de uma iteração, a invariante se mantém.

**Término:** O laço é finalizado quando $i > n$, ou seja, $i = n + 1$. Substituindo $i$ por $n + 1$ na invariante, temos que: $fatorial = 1 \times 2 \times ... \times (n+1) - 1$, ou seja, $fatorial = 1 \times 2 \times ... \times n$, que corresponde a $n!$. Logo, o algoritmo está correto.

## Exemplo 3: Potência

```
1: function POTENCIA(x,n)
2:     p=1
3:     i=0
4:     while i < n do
5:         p = p * x
6:         i = i + 1
7:     end while
8:     return p
9: end function
```

**Invariante:**

$p = x^i$

ou de forma textual:

*Antes do início de cada iteração do* `while`*, p é o resultado da multiplicação de x por si mesmo i vezes.*

**Prova de correção por invariante de laço:**

**Inicialização:** Antes da inicialização $i = 0$ e $p = 1$, logo $p = x^0$ que, independente do valor do x, é 1. Assim, a invariante é verdadeira na inicialização.

**Manutenção:** Supondo que a invariante está correta, ou seja, $p = x^i$, $p$ recebe o valor da multiplicação de $x$ por $p$ (linha 5), em seguida, $i$ é incrementado em 1 (linha 6). Verificando a invariante temos que $p = x^i * x$, o que resulta em $p = x^{i+1}$. Como i é incrementado, $p = x^i$, o que mostra que depois de uma iteração, a invariante se mantém.

**Término:** O laço é finalizado quando $i \geq n$, ou seja, $i = n$. Substituindo $i$ por $n$ na invariante temos $p = x^n$, que corresponde a potência de $x$ elevado a $n$. Logo, o algoritmo está correto.

## Exemplo 4: Maior elemento de vetor

```
1: function MaiorElemento(V[],n)
2:     max=V[1]
3:     for (i = 2; i ≤ n; i++) do
4:         if V[i] > max then
5:             max = V[i]
6:         end if
7:     end for
8:     return max
9: end function
```

**Invariante:**

*Antes do início de cada iteração do* `for`*, max contém o maior valor do subvetor* $V[1..i-1]$.

**Prova de correção por invariante de laço:**

**Inicialização:** Antes da inicialização $i = 2$ e $max = V[1]$, logo o subvetor $V[1..i-1]$ corresponde a $V[1..2-1] \rightarrow V[1..1]$. Assim, a invariante é verdadeira na inicialização, pois em um vetor com apenas um elemento, o elemento na primeira posição é o maior.

**Manutenção:** Supondo que a invariante está correta, ou seja, $max$ contém o maior valor do subvetor $V[1..i-1]$, na linha 4, temos duas situações: (a) se condição `verdadeira`, $V[i] > max$, logo $max = V[i]$ (linha 5) e, em seguida, i é incrementado (for). Verificando a invariante temos que $max$ armazena o último elemento do subvetor $V[1..i]$ e como i é incrementado, $max$ possui o maior valor do subvetor $V[1..i-1]$; (b) se condição `falsa`, i apenas será incrementado (for), e $max$ ainda armazenará o maior elemento do vetor $V[1..i-1]$. Logo, considerando as duas possíveis iterações, a invariante se mantém.

**Término:** O laço é finalizado quando $i > n$, ou seja, $i = n+1$. Substituindo $i$ por $n+1$ na invariante, temos o subvetor $V = V[1..(n+1)-1] = V[1..n]$ e o maior valor em $max$. Como o subvetor corresponde ao vetor original, o algoritmo está correto.

## Exemplo 5: Bubblesort

Quando temos laços aninhados, devemos analisar os laços individualmente começando pelo laço mais interno. Para cada laço é necessário especificar um invariante e realizar a prova. O resultado do término do laço interno é usado na prova do próximo laço.

```
1: function BUBLESORT(V[],n)
2:     for (i = 1; i ≤ n; i++) do
3:         for (j = n; j ≥ i + 1; j– –) do
4:             if V[j] < V[j − 1] then
5:                 troca(V[j], V[j − 1])
6:             end if
7:         end for
8:     end for
9: end function
```

**Invariante - Laço interno (linhas 3-6):**

*Antes do início de cada iteração do* `for`*, o elemento A[j] é o menor elemento do subvetor* $V[j..n]$.

**Inicialização:** Antes da inicialização $j = n$, logo o subvetor $V[j..n]$ corresponde a $V[n..n]$. Assim, a invariante é verdadeira na inicialização, pois $A[j]$ é o menor elemento do vetor.

**Manutenção:** Supondo que a invariante está correta, ou seja, o elemento A[j] é o menor elemento do subvetor $V[j..n]$, na linha 4, temos duas situações: (a) se condição `verdadeira`, $V[j] < V[j-1]$, em seguida, os valores são permutados e $j$ é decrementado (for). Verificando a invariante, temos que o menor elemento está em $j - 1$ devido a permutação, mas como o $j$ é decrementado, o menor elemento estará em $V[j]$ para o subvetor $V[j..n]$; (b) se condição `falsa`, o menor elemento está em $V[j-1]$ e $j$ será decrementado (for), logo o menor elemento estará em $V[j]$ para o subvetor $V[j..n]$. Considerando as duas possíveis iterações, a invariante se mantém.

**Término:** O laço é finalizado quando $j < i + 1$, ou seja, $j = i$. Substituindo na invariante, temos que $V[i]$ é o menor elemento do subvetor $V[i..n]$.

**Invariante - Laço externo (linhas 2-8):**

*Antes do início de cada iteração do* `for`*, o subvetor* $V[1..i-1]$ *está ordenado.*

**Inicialização:** Antes da inicialização $i = 1$, logo o subvetor $V[1..i-1]$ corresponde a $V[1..0]$. Assim, a invariante é verdadeira na inicialização, pois não há elementos no subvetor, assim consideramos ordenado.

**Manutenção:** Supondo que a invariante está correta, ou seja, subvetor $V[1..i-1]$ está ordenado, e, como o laço interno garante que $V[i]$ é o menor elemento de $V[i..n]$ e, em seguida, i é incrementado, os elementos de $V[1..i-1]$ continuam ordenados (importante notar que todos os elementos $V[1..i-1]$ são menores aos elementos de $V[i..n]$ devido à prova do laço interno), logo a invariante se mantém.

**Término:** O laço é finalizado quando $i > n$, ou seja, $i = n + 1$. Substituindo $i$ por $n + 1$ na invariante, temos que $V[1..i-1] \rightarrow V[1..(n+1)-1] \rightarrow V[1..n]$ está ordenado. Logo, o algoritmo está correto.

# Bibliografia

[CRLS] CORMEN, T. H. et al. Algoritmos: Teoria e Prática. Elsevier, 2012. 3a Ed. Capítulo 2 (Dando a Partida), Seção 2.1 (Ordenação por Inserção)