
CORREÇÃO DE PROGRAMAS - Aula 2 (Cormen, §2.1)
---

1. DEFINIÇÃO: estes são dois tipos de análise de correção de algoritmos:

   * CORREÇÃO PARCIAL: garantir que o algoritmo retorna uma "resposta" correta
     CASO ele termine de executar.

   * CORREÇÃO TOTAL: garantir que o algoritmo é parcialmente correto E QUE ELE
     SEMPRE TERMINA DE EXECUTAR.

   Naturalmente, essa classificação faz mais sentido para algoritmos cuja
   finalidade é a computação de algum valor. (Em contraste, há programas que
   realizam a tarefa desejada antes de chegarem ao fim, por serem responsáveis
   por prover algum serviço: sistemas operacionais, etc.)

   Mais detalhes na Wikipédia (referência ao final).

USANDO INVARIANTES PARA VERIFICAR ALGORITMOS

Veja o algoritmo de busca linear anotado com invariantes:

```
  ========================================================
  Algoritmo: busca_linear
  Entrada: um vetor A[1..n] e um valor "v".
  Saída: um número natural.
  --------------------------------------------------------
  1. i := 1
     // INVARIANTES: * 1 <= i <= n+1
     //              * ∀j ∈ {1..i-1}, A[j] != v
  2. ENQUANTO i <= n
  3. | SE A[i] = v
  4. | | RETORNE i.
  5. | ++i
  6. RETORNE i.
  ========================================================
```

Antes de nos determos na questão de como provar esses invariantes (isto é,
provar que as afirmações acima de fato são invariantes), vejamos como nós
podemos utilizá-los para mostrar que o programa acima está PARCIALMENTE
CORRETO:

(OBSERVAÇÃO: a especificação de correção parcial abaixo não está completa, pois
 não inclui, por exemplo, a exigência de que o algoritmo não modifique o vetor.
 Uma especificação mais detalhada poderia ser levada à frente, mas nós não
 faremos isso, pois o nível de detalhe abaixo é suficiente para ilustrar o uso
 de invariantes.)

2. LEMA: se os invariantes do algoritmo estão corretos, então, se uma chamada busca_linear(A,v) retorna um valor "k", então:

* ou k ∈ {1..n} e A[k] é a primeira ocorrência de "v" em "A",
* ou k = n+1 e "v" não ocorre em "A".

==========
PROVA:

Considere uma chamada busca_linear(A,v) que retorna um número natural "k". Os casos possíveis são os seguintes:

1. O retorno acontece pela linha 4: nesse caso, considere a iteração do laço durante a qual o algoritmo retorna, e seja "x" o valor de "i" no início dessa iteração. Pelo 1º invariante, temos que, no início da iteração, 1 <= i. Além disso, como a iteração é executada, então a condição da linha 2 é verdadeira, e portanto i <= n. Logo, 1 <= x <= n. Além disso, pelo 2º invariante do laço, temos que, no início da iteração em questão,
   ∀j ∈ {1..x-1}, A[j] != v.
   Adicionalmente, como o algoritmo retorna durante essa iteração, então a condição da linha 3 é verdadeira, e portanto A[x] = v. Logo, como k = x, então k ∈ {1..n} e A[k] é a primeira ocorrência de "v" em "A".

2. O retorno acontece pela linha 6: nesse caso, pela forma do algoritmo, o laço é executado, chega ao fim e o valor "k" de "i" é então retornado. Pelo 1º invariante do laço, temos que, ao fim da execução do laço, 1 <= i <= n+1. Além disso, como a condição da linha 2 é falsa ao fim do laço, temos i > n. Logo, temos então i = n+1. Além disso, pelo 2º invariante do laço, temos
   ∀j ∈ {1..(n+1)-1}, A[j] != v.
   Logo, "v" não ocorre em "A", e o algoritmo retorna "n+1".

Pela análise de casos acima, portanto, o enunciado é verdadeiro, CQD.
==========

PROVANDO INVARIANTES DE LAÇO

Nesse momento, deve estar evidente a utilidade do conceito "invariante de laço". Resta, porém, saber como provar que uma afirmação é de fato um invariante de laço. O processo é essencialmente uma indução sobre as iterações do laço e consiste em dois passos:

1. VALIDADE INICIAL: mostrar que a afirmação é válida no início do laço, isto é, logo antes de o laço começar.

2. MANUTENÇÃO: mostrar que a afirmação continua verdadeira ao fim de todas as iterações da execução do laço. Para tanto, argumenta-se de forma semelhante ao passo de uma indução:

   a) considera-se uma iteração qualquer que chega ao fim (**), isto é, uma iteração ao final da qual a condição do laço é novamente avaliada,
   b) supõe-se que a afirmação é válida no início da iteração, e
   c) prova-se que a afirmação continua verdadeira ao fim da iteração.

   **: algumas iterações não chegam a ser concluídas, por exemplo quando um comando de retorno é executado durante a iteração. Tais iterações não são incluídas na prova da manutenção de um invariante de laço (pelo menos não no sentido acima; naturalmente, numa prova de manutenção de invariante, o fato de que a iteração em consideração chega ao fim é frequentemente utilizado; veja os exemplos abaixo).

Essa metodologia é ilustrada a seguir:

3. LEMA: os invariantes do algoritmo busca_linear estão corretos.

==========
PROVA:

Invariante "1 <= i <= n+1":

  * VALIDADE INICIAL: É verdade no início do laço, pois então temos i = 1 e
    ---------------- n >= 0.

  * MANUTENÇÃO: Considere uma iteração qualquer do laço, ao final da qual o
    ---------- algoritmo novamente vai ao teste da linha 2. Logo, i <= n no
    início da iteração (pela condição do laço), e além disso a linha 5 é
    executada durante a iteração (pois a linha 4 não o é). Logo, temos i <= n+1
    ao fim da iteração, CQD.

    (Mais precisamente, seja "x" o valor de "i" no início da iteração; logo,
     x <= n. Além disso, "i" vale "x+1" ao fim da iteração. Como x+1 <= n+1,
     então "i <= n+1" vale no início da iteração. Além disso, como 1 <= x+1,
     então "1 <= i" também vale ao fim da iteração, CQD.)

Invariante "∀j ∈ {1..i-1}, A[j] != v": (∀j ∈ N, se 1 <= j < i, então A[j] != v.)

  * VALIDADE INICIAL: É verdade no início do laço, pois i=1 e ∄j ∈ {1..i-1} = ∅.
    ----

  * MANUTENÇÃO: Considere uma iteração qualquer do laço, ao final da qual o
    ---------- teste da linha 2 volta a ser executado. Seja "x" o valor de "i"
    no início da iteração. Logo, como a afirmação é verdadeira no início da
    iteração, então ∀j ∈ {1..x-1}, A[j] != v. Além disso, como a iteração chega
    ao fim, então a condição da linha 3 é avaliada falsa, e portanto A[x] != v.
    Logo, ∀j ∈ {1..x}, A[j] != v. Finalmente, como a linha 5 é executada, então
    "i" vale "x+1" ao fim da iteração, e portanto a afirmação é verdadeira ao
    fim da iteração, CQD.

==========

4. EXERCÍCIOS: Prove a correção parcial de:

   a) Um algoritmo para calcular o maior elemento de um vetor de números reais.

   b) O algoritmo de MDC de Euclides. (RESPOSTA: CAnA 2014-2, AP1, Q1.)

5. REFERÊNCIAS:

   a) Sobre correção parcial e total:

      http://en.wikipedia.org/wiki/Correctness_%28computer_science%29