Prova de Correção por Invariante de Laço

Prof. Juliano Foleis

Prova de Correção por Invariante de Laço

- Uma tarefa importante relativa a análise de algoritmos é a prova de correção (ou corretude).
- Consiste em mostrar que um algoritmo está correto.

Definição: Algoritmo Correto

Um algoritmo é considerado correto quando ele respeita a relação ***entrada*** $\rightarrow$ ***saída*** para todas as entradas possíveis que respeitem as pré-condições impostas.

Algoritmos Corretos - Exemplos

1. Um algoritmo de ordenação que impõe a pós-condição de ordenação em qualquer vetor que receba de entrada, seja ele um vetor unitário, um vetor vazio, um vetor arbitrário, ou um vetor contendo apenas elementos repetidos.
2. Um algoritmo de busca sequencial em vetor que retorna a posição correta que um elemento ocupa o vetor, ou -1 se o elemento não ocupa o vetor. Isto deve funcionar em vetores unitários, vazios, um vetor qualquer, ou em um vetor contendo apenas elementos repetidos.

Algoritmos Corretos - Exemplos

3. Um algoritmo de busca binária em vetor que retorna a posição correta que um elemento ocupa o vetor, ou -1 se o elemento não ocupa o vetor. Uma pré-condição (propriedade que deve ser verdadeira) é que o vetor de entrada esteja ORDENADO. Isto deve funcionar em vetores unitários, vazios, um vetor ordenado qualquer, ou em um vetor contendo apenas elementos repetidos.

Como provar que um algoritmo está correto?

- Para a grande maioria dos algoritmos, não é possível provar que ele está correto executando-o em todas as entradas possíveis.
  - Nestes algoritmos existem infinitas entradas possíveis;
  - Pela grande (ou infinita) quantidade de entradas possíveis, seria impossível executar o algoritmo todas as vezes necessárias!

Existem muitas técnicas para provar que um algoritmo está correto.

Prova de Correção por Invariante de Laço

Vamos estudar o método de prova de correção por invariante de laço.

- Teoricamente baseia-se na indução matemática;
- É construída a partir de um argumento lógico-matemático;
- Serve apenas para provar que algoritmos iterativos estão corretos.

Prova de Correção por Invariante de Laço

A correção por invariante de laço serve para provar 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 do laço seja relevante na relação entrada-saída do problema resolvido pelo algoritmo.

Prova de Correção por Invariante de Laço

O método consiste em três passos, que equivalem a três momentos da execução de um laço de repetição:

1. **Inicialização:** A invariante deve ser verdadeira antes da primeira iteração de um laço, logo após os procedimentos de inicialização;

2. **Manutenção:** ***Supondo* que a invariante é verdadeira antes de uma iteração qualquer do laço**, devemos mostrar que o que acontece na execução do laço mantém a invariante verdadeira antes da próxima iteração; e

3. **Término:** Quando a execução do laço termina, a invariante é verdadeira e pode ser utilizada como hipótese para provar que o algoritmo está correto.

Exemplo 1

```
int soma_elementos(int* v, int n){
  int soma = 0;
  int i = 0;
  while(i < n){
    soma = soma + v[i];
    i = i + 1;
  }
  return soma;
}
```

Invariante:

Antes do início de cada iteração do laço `while`, soma = v[0] + v[1] + … + V[i-1].

Este algoritmo está correto somente se retornar a soma de todos elementos do vetor `v` de tamanho `n`, para qualquer vetor `v` de inteiros e n ≥ 0.

Exemplo 1

```c
1 int soma_elementos(int* v, int n){
2   int soma = 0;
3   int i = 0;
4   while(i < n){
5     soma = soma + v[i];
6     i = i + 1;
7   }
8   return soma;
9 }
```

Invariante:

Antes do início de cada iteração do laço `while`, soma = v[0] + v[1] + … + V[i-1].

Inicialização: Logo antes da primeira iteração do laço while, `soma=0` e `i=0`. A invariante diz que: soma = v[0] + v[1] + … + v[i-1]. Como `soma=0` e `i=0`,

$$\begin{aligned} 0 &= v[0] + \ldots + v[0-1] \\ 0 &= v[0] + \ldots + v[-1] \\ 0 &= 0 \text{ (pq o vetor v[0..-1] é vazio!)} \end{aligned}$$

Como `v[0..-1]` é um vetor vazio, a soma v[0] + v[1] + … + v[i-1] é 0, que é o valor atribuído a `soma` na linha 2. Portanto, a invariante de laço é verdadeira antes da primeira iteração do `while`.

Exemplo 1

```
1 int soma_elementos(int* v, int n){
2   int soma = 0;
3   int i = 0;
4   while(i < n){
5     soma = soma + v[i];
6     i = i + 1;
7   }
8   return soma;
9 }
```

Invariante:

Antes do início de cada iteração do laço `while`, `soma = v[0] + v[1] + ... + V[i-1]`.

Manutenção: Vamos supor que a invariante é verdadeira no início de uma iteração `i` qualquer. Neste caso, sabemos que soma = v[0] + v[1] + v[2] + … + v[i-1].

após a linha 5,

soma = v[0] + v[1] + v[2] + … + v[i-1] + v[i]

após a linha 6,

soma = v[0] + v[1] + v[2] + … + v[i-2] + v[i-1]

Portanto, ao final da iteração `i` e logo antes da iteração `i+1`, a invariante continua verdadeira. Como o argumento é para um $`i` qualquer, a invariante é verdadeira entre duas iterações quaisquer.

Exemplo 1

```
int soma_elementos(int* v, int n){
  int soma = 0;
  int i = 0;
  while(i < n){
    soma = soma + v[i];
    i = i + 1;
  }
  return soma;
}
```

Invariante:

Antes do início de cada iteração do laço `while`, `soma = v[0] + v[1] + ... + V[i-1]`.

Término: Depois da última iteração do laço `while`, `i=n`. Como provamos que a invariante é mantida entre todas as iterações, sabemos que: soma = v[0] + v[1] +...+ v[i-1]. Substituindo `i` por `n+1`, que é o valor atual de `i`,

$$\text{soma} = v[0] + v[1] + \ldots + v[n-1]$$

Desta forma, o valor de `soma` corresponde à soma de todos os elementos do vetor. Portanto, a variável `soma` é retornada como resultado final, contendo a soma de todos os elementos do vetor, que é o que queríamos mostrar que o algoritmo faz. Portanto, o algoritmo `soma_elementos` 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)