Análise e Técnicas de Algoritmos

Jorge Figueiredo

Corretude de Algoritmos

Agenda

- Indução Matemática
- Corretude de Algoritmos Recursivos
- Invariante de Laços
- Corretude de Algoritmos Não-Recursivos

Indução Matemática

- Técnica de prova poderosa e comumente usada em Ciência da Computação.
- Axioma da Indução:
  - Suponha que:
    - **P(0)** é true
    - $\forall m \in N$, $P(m) \rightarrow P(m+1)$
  - Então:
    - $\forall n \in N$, P(n) é true

Formato da Prova por Indução

- O texto de uma prova por indução consiste de três partes:
  - Especificação da hipótese da indução:
    - P(n)
  - O caso base:
    - P(0)
  - O passo indutivo:
    - $\forall m \in N$, $P(m) \rightarrow P(m+1)$

Exemplo: Identidade de Gauss

- Queremos provar que: $1 + 2 + \ldots + n = n.(n+1)/2$
- Hipótese da Indução (H.I.):
  - A própria identidade
  - $I(n) = n.(n + 1)/2$
- Caso base:
  - $n = 1$
  - $I(1) = 1 = 1.(1 + 1)/2 = 1$
- Passo indutivo:
  - Provar que $I(n + 1) = (n + 1).(n + 2)/2$
  - Utilizar a H.I.

Variações

- Variação 1
  - Caso base: $n = 1$
  - Provar que para $\forall n \geq 2$, se a propriedade é válida para n-1, ela é válida para n.
- Variação 2
  - Vários casos base: n = 1, 2 e 3
  - Provar que para $\forall n \geq 4$, se a propriedade é válida para n, ela é válida para n+1.
- Variação 3 (indução forte)
  - Caso base: $n = 1$
  - Provar que para $\forall n \geq 2$, se a propriedade é válida para $\forall 1 \leq m \leq n$, ela é válida para n+1.

Árvore Binária Completa

- Provar que uma árvore binária completa com k níveis tem exatamente $2^k - 1$ nós.
- Prova:
  - Caso base: k=1. Ok, pois nesse caso tem um único có.
  - H.I.: $S(k) = 2^k - 1$ nós.
  - Mostrar que $S(k+1) = 2^{k+1} - 1$ nós.
    - Sabe-se que $S(K+1) = 2.S(k) + 1$

Triomino

- Tabuleiro mxm, em que m é potência de 2, ou seja, $m = 2^n$.
- Uma das células do tabuleiro é a célula especial.
- Uma L-peça é semelhante a um tabuleiro 2x2, eliminando-se uma das células.
- É possível cobrir com L-peças todo o tabuleiro, com exceçãoda célula especial?

Corretude de Algoritmos Recursivos

- Provar usando indução.
- Caso base é o caso base da recursão.
- Assumir que as chamadas recursivas estão corretas, e usar esse argumento para provar que a execução corrente é correta (passo indutivo).

Corretude de Fibonnaci Recursivo

$F_0 = 0$, $F_1 = 1$, $\forall\ n \geq 2$, $F_n = F_{n-1} + F_{n-2}$

```
Fib(n)
if  n≤1 then
      return n
            else
      return Fib(n – 1) + Fib(n – 2)
```


Corretude de Algoritmo Recursivo

Encontrar o maior valor em um array A de n elementos.

```
Maximo(n)
if  n≤1 then
      return A[1]
            else
      return max(Maximo(n – 1), A[n])
```


Invariante de Laço

- Conceito importante que pode nos ajudar a analisar programas.
- Pode ser definido como uma relação entre as variáveis de um algoritmo (programa) que é verdadeira nas seguintes condições:
  - Antes do início do laço
  - Durante a manutenção do laço.
  - Na saída do laço.

Exemplo de Invariante de Laço

Algoritmo que calcula a soma de todos os elementos de um array

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

$\forall j \geq 0,\ i_j = j + 1,\ s_j = A[1] + \ldots + A[j]$

Ainda sobre Laços

- Dois elementos:
  - Guarda: expressão booleana que indica se o corpo do laço deve ou não ser executado.
  - Variante: expressão numérica que mede o quanto de execução ainda falta.

```
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:
  1. Início: invariante é verdadeiro antes de entrar no laço.
  2. Invariância: o corpo do laço preserva o invariante.
  3. Progresso: o corpo do laço diminui a variante.
  4. Limitação: quando a variante atinge certo valor, a guarda se torna falsa.
  5. Saída: a negação da guarda e o invariante descrevem o objetivo do laço.

Derivando Laços a partir de Invariante

```
PRE
►invariante verdadeiro
while G do
      ►invariante verdadeiro
      C
      ►invariante verdadeiro
►¬G e invariante verdadeiro
POS
```


Exemplo: Derivando Laços a partir de Invariante

Considerar a definição da divisão de inteiros que relaciona as saídas q (quociente) e r (resto) às entradas n (número) e d (divisor). Por definição: $r < d$ e $n = q \times d + r$.

- A definição da divisão corresponde a pós-condição do laço:
  - $\neg G$ é $r < d$ e Invariante é $n = q \times d + r$.
  - G é $r \geq d$.

Exemplo: Derivando Laços a partir de Invariante

```
PRE
► n = q × d + r
while r≥d do
      ► n = q × d + r
      C
      ► n = q × d + r
►¬(r≥d) e n = q × d + r
POS
```

```
r ← n
q ← 0
► n = q × d + r
while r≥d do
      ► n = q × d + r
      r ← r – d
      q ← q + 1
      ► n = q × d + r
►¬(r≥d) e n = q × d + r
return q
```

Corretude de Algoritmos Não Recursivos

- Analisar um laço por vez, começando pelo mais interno se houver aninhamento de laços.
- Para cada laço determinar um invariante de laço.
- 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.
- Restringir a algoritmos com um laço:
  - O valor do identificador *x* após a *i*-ésima iteração de um laço é x_i ($i=0$ indica o valor de *x* imediatamente antes de iniciar o laço).

Corretude de Fibonnaci Não Recursivo

$F_0 = 0$, $F_1 = 1$, $\forall\, n \geq 2$, $F_n = F_{n-1} + F_{n-2}$

```
Fib(n)
if  n = 0 then
      return 0
else
      a = 0
      b = 1
      i = 2
      while i ≤ n do
            c = a + b
            a = b
            b = c
            i = i + 1
return b
```


Corretude de Fibonnaci Não Recursivo

Fatos sobre o algoritmo

```
Fib(n)
if  n = 0 then
      return 0
else
      a = 0
      b = 1
      i = 2
      while i ≤ n do
            c = a + b
            a = b
            b = c
            i = i + 1
return b
```

$i_0 = 2$
$i_{j+1} = i_j + 1$
$a_0 = 0$
$a_{j+1} = b_j$
$b_0 = 1$
$b_{j+1} = c_{j+1}$
$c_{j+1} = a_j + b_j$

Corretude de Fibonnaci Não Recursivo

Invariante de laço

```
Fib(n)
if  n = 0 then
      return 0
else
      a = 0
      b = 1
      i = 2
      while i ≤ n do
            c = a + b
            a = b
            b = c
            i = i + 1
return b
```

$\forall j \geq 0$, $i_j = j + 2$, $a_j = F_j$ e $b_j = F_{j+1}$

Prova por indução

Caso base: $j = 0$
Hipótese da Indução: próprio invariante
Prova p/ j+1: usando os fatos e a H.I.

Corretude de Fibonnaci Não Recursivo

Mostrar que termina e retorna o valor correto ($b = F_n$)

```
Fib(n)
if  n = 0 then
      return 0
else
      a = 0
      b = 1
      i = 2
      while i ≤ n do
            c = a + b
            a = b
            b = c
            i = i + 1
return b
```

Prova

- para n = 0, OK.
- Como $i_{j+1} = i_j + 1$, eventualmente $i = n+1$
- Assumir que ocorre depois de t iterações: $i_t = n+1$
- $i_t = t + 2$, logo: $t = n - 1$
- Pelo invariante, $b_t = F_{t+1}$.
- Logo: $b_t = F_n$

Exercício

Mostrar que o algoritmo abaixo computa x^n.

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