Fontes principais

1. Cormem T. H.; Leiserson C. E.; Rivest R.: Stein C.. *Introduction to Algorithms*, 3^a edição, MIT Press, 2009

2. Análise de algoritmo - IME/USP (prof. Paulo Feofiloff) **http://www.ime.usp.br/~pf/analise_de_algoritmos**

Princípio de indução matemática

1^o princípio de indução matemática (fraco)

A prova de uma afirmação por indução matemática é feita em dois passos:

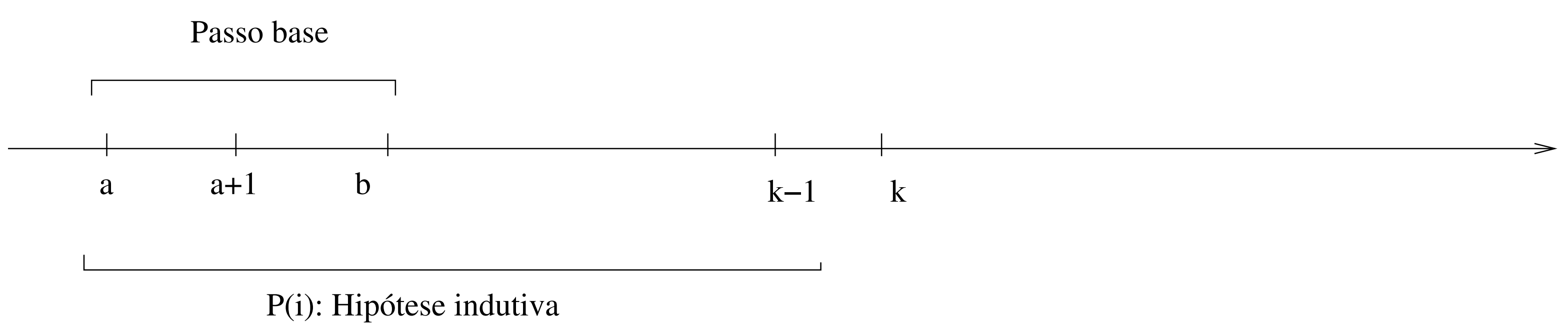
▷ (1) Passo base: É provado $P(n_0)$ é verdade para um dado específico.

▷ (2) Passo indutivo: É provado para todos os valores $k \geq n_0$, se $P(k)$ é verdade então $P(k+1)$ é verdade.

2^o princípio de indução matemática (forte)

Seja $P(n)$ um predicado que é definido para o inteiro n, e seja a e b inteiros fixos, com $a \leq b$.

A prova de $P(n)$ consiste em verificar a veracidade das seguintes afirmações:
▷ (1) $P(a), P(a+1), \cdots, P(b)$ são verdades (Passo base)
▷ (2) Para qualquer $k \geq b$, se $P(i)$ é verdade para $a \leq i < k$, então $P(k)$ é verdade (Passo indutivo).



Corretude de algoritmos iterativos

Corretude de algoritmos iterativos

Definição 1. *Um **invariante** de um laço é uma propriedade que relaciona várias variáveis de um algoritmo a cada execução completa daquele laço.*

Corretude de algoritmos iterativos

Estratégia "típica" para mostrar a corretude de um algoritmo iterativo através de invariantes:

▷ (1) Mostre que o invariante vale no início da primeira iteração (trivial, em geral).
▷ (2) Suponha que o invariante vale no início de uma iteração qualquer e prove que ele vale no início da próxima iteração.

Corretude de algoritmos iterativos

▷ (3) Conclua se o algoritmo pára e o invariante vale no início da última iteração, então o algoritmo está **correto**.

Note que (1) e (2) implicam que o invariante vale no início de qualquer iteração do algoritmo. Isto corresponde ao método de **indução matemática**.

Algoritmo para somar n elementos

Soma-Vetor(A, n)

```
1  s ← 0
2  para j ← 1 até n
3      faça s ← s + A[j]
4  retorne s
```

Corretude do algoritmo. Invariante na linha 2.

▷ No início de cada iteração j, s contém a soma dos elementos da posição 1 até $j-1$, portanto vale que $s = \sum_{i=1}^{j-1} A[i]$

Algoritmo para somar n elementos

Corretude do algoritmo. Invariante na linha 2.

$\triangleright$ Na primeira iteração temos que $j = 1$ e portanto $s = 0 = \sum_{i=1}^{0} A[i]$. Ou seja o invariante vale.

Algoritmo para somar n elementos

Corretude do algoritmo. Invariante na linha 2.

$\triangleright$ Suponha que no início da iteração j o invariante vale, ou seja, $s = \sum_{i=1}^{j-1} A[i]$.

Então o algoritmo adiciona $A[j]$ a s e portanto, no início da iteração $j+1$ temos que $s = \sum_{i=1}^{j} A[i]$. Este é exatamente o invariante na iteração $j+1$

Algoritmo para somar n elementos

Na última iteração temos que $j = n + 1$ (laço pára) e a correção do algoritmo é evidente, pois o invariante diz que

$$s = \sum_{i=1}^{n} A[i]$$

Algoritmo para calcular o fatorial de n

Algoritmo para calcular o fatorial de n

Fatorial(n)

1 $fat \leftarrow 1$
2 $i \leftarrow 1$
3 **enquanto** $(i \leq n)$
4 **faça** $fat \leftarrow fat * i$
5 $i \leftarrow i + 1$
6 **retorne** fat

Qual é o invariante na linha 3?

Fatorial(n)

1 $fat \leftarrow 1$
2 $i \leftarrow 1$
3 **enquanto** $(i \leq n)$
4 **faça** $fat \leftarrow fat * i$
5 $i \leftarrow i + 1$
6 **retorne** fat

Qual é o invariante na linha 3?

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

No início de cada iteração i, fat contém o valor calculado do fatorial de 1 até $i - 1$.

Algoritmo para calcular o fatorial de n

▷ Início

Antes do início do laço $i = 1$, assim, $fat = 1$. Isto mostra que o invariante está correto no início da primeira iteração.

Algoritmo para calcular o fatorial de n

▷ Invariância (durante a execução do laço)

Suponha que o invariante está correto no início da iteração i, isto é, $fat = \prod_{k=1}^{i-1} k$.

O algoritmo multiplica este valor por i, obtendo $\prod_{k=1}^{i} k$, e logo após incrementa i de 1. Portanto, isto mostra que depois da iteração o invariante se mantém.

Algoritmo para calcular o fatorial de n

▷ Término

O laço termina quando $i > n$, isto é, $i = n + 1$. Substituindo i por $n + 1$ no invariante, temos que:

$$fat = \prod_{k=1}^{n} k = 1 * 2 * \cdots * n = n!$$

Portanto, o algoritmo está correto.

Laços aninhados

Laços aninhados

▷ Analisar um laço por vez começando pelo mais interno.

▷ Para cada laço, determinar um invariante.

▷ Provar que o invariante é válido.

▷ Mostrar que o algoritmo termina.

▷ Usar o invariante para provar que o algoritmo retorna o valor desejado.

Análise do algoritmo: Insertion sort

Análise do algoritmo: Insertion sort

Insertion-Sort(A, n)

para $j \leftarrow 2$ **até** n
    faça $chave \leftarrow A[j]$
        $\triangleright$ Insere $A[j]$ na sequência ordenada $A[1 \,..\, j-1]$.
        $i \leftarrow j - 1$
        enquanto $i > 0$ **e** $A[i] > chave$
            faça $A[i+1] \leftarrow A[i]$
                $i \leftarrow i - 1$
        $A[i+1] \leftarrow chave$

Análise do algoritmo: Insertion sort

▷ Invariante principal (i_1):

No começo de cada iteração do laço da linha (1) o subvetor $A[1..j-1]$ está ordenado.

1						j				n
20	25	35	40	44	55	38	99	10	65	50

∘ Suponha que o invariante é válido.

∘ Então a corretude do algoritmo é "evidente".

∘ No início da última iteração temos $j = n + 1$. Assim, do invariante segue que o subvetor $A[1..n]$ está ordenado.

Análise do algoritmo: Insertion sort

$\triangleright$ Invariantes auxiliares:

No início da linha 5 valem os seguintes invariantes:

(i_2): $A[1..i]$ e $A[i+2..j]$ contém elementos de $A[1..j]$ antes de entrar no laço da linha 5.
(i_3): $A[1..i]$ e $A[i+2..j]$ são crescentes
(i_4): $A[1..i] \leq A[i+2..j]$
(i_5): $A[i+2..j] > chave$

Análise do algoritmo: Insertion sort

Invariante forte (i'_1):

No começo de cada iteração do laço da linha 1, o subvetor $A[1..j-1]$ é uma permutação ordenada do subvetor original $A[1..j-1]$.

$$\left.\begin{array}{l}\text{Invariantes } (i_2) \text{ a } (i_5)\\ +\ \text{condição de parada na linha 5}\\ +\ \text{atribuição da linha 7.}\end{array}\right\} \Longrightarrow (i'_1)$$

Análise do algoritmo: Insertion sort

Esboço da demonstração de (i'_1):

Validade na primeira iteração: neste caso, temos $j = 2$ e o invariante simplesmente afirma que $A[1..1]$ está ordenado, o que é óbvio.

Análise do algoritmo: Insertion sort

Validade na iteração $j > 2$: segue da discussão anterior, os elementos maiores que a chave são "empurrados" para seus lugares corretos e a chave é colocada no "espaço vazio".

Mas, uma demonstração mais formal deste fato exigiria uma prova dos invariantes auxiliares do laço interno.

Análise do algoritmo: Insertion sort

Na última iteração: temos $j = n+1$ e logo $A[1..n]$ está ordenado.

Portanto, o algoritmo está correto.

Corretude de algoritmos recursivos

Corretude de algoritmos recursivos

Em muitos problemas computacionais cada solução de uma instância do problema contém soluções de instâncias menores.

Por exemplo: Divisão e conquista.

Corretude de algoritmos recursivos

Corretude provada na indução:

- Passo base é a base da recursão

- Assumir que as chamadas recursivas estão corretas, e usar tal argumento para provar que a execução corrente está correta (passo indutivo)

Algoritmo recursivo para somar n elementos

Algoritmo recursivo para somar n elementos

Soma-Vetor(A, n)

se $n = 0$
    então $s \leftarrow 0$
    senão $s \leftarrow$ Soma-Vetor$(A, n - 1) + A[n]$
retorne s

Vamos provar a corretude por indução.

Algoritmo recursivo para somar n elementos

- Passo base:

  $n = 0$ (vetor sem nenhum elemento), $s = 0$ (trivial)

- Passo indutivo:

  Hipótese: Assumimos que a chamada Soma-Vetor$(A, n-1)$ está correta. Se Soma-Vetor$(A, n-1)$ está correto, então para todo $n > 0$ o algoritmo retornará a soma dos $n-1$ elementos mais $A[n]$.

Portanto, o algoritmo está correto.

Algoritmo recursivo de Fibonacci

Algoritmo recursivo de Fibonacci

Fib(n)

1 **se** $n \leq 1$

2 **então retorne** n

3 **senão retorne** Fib($n-1$) + Fib($n-2$)

Vamos provar a corretude por indução.

Algoritmo recursivo de Fibonacci

- Passo base:

  Para $n = 0$ temos Fib$(0) = 0$.
  Para $n = 1$ temos Fib$(1) = 1$.

- Passo indutivo:

  Hipótese: Para $n \geq 2$ e para todo $0 \leq m < n$, as chamadas de Fib$(n-1)$ e Fib$(n-2)$ estão corretas.

  Assim temos:

  $$\text{Fib}(n-1) + \text{Fib}(n-2) \underset{def}{\overset{hip}{=}} \text{Fib}(n)$$

  Portanto, o algoritmo está correto.

Obrigado