MC358 - Fundamentos matemáticos da computação

Prof. Dr. Hilder Vitor Lima Pereira

Instituto de computação

UNICAMP

04 de setembro de 2023

1. Revisão sobre o princípio da indução matemática

2. Princípio da Indução Completa: continuação

3. Provando corretude de algoritmos

4. Perguntas, observações, comentários?

Revisão sobre o princípio da indução matemática

Princípio da indução matemática (PIM)

Funcionamento:

- Provamos o caso base $P(n_0)$.
- Assumimos $P(k)$ para algum $k \geq n_0$.
- Deduzimos $P(k+1)$ (ou seja, provamos $P(k) \Rightarrow P(k+1)$).

Funcionamento:

- Provamos o caso base $P(n_0)$.
- Assumimos $P(k)$ para algum $k \geq n_0$.
- Deduzimos $P(k+1)$ (ou seja, provamos $P(k) \Rightarrow P(k+1)$).

Podemos usar

- incremento $m > 1$ no passo indutivo, mas devemos provar m casos base;
- $P(k-1)$ como hipótese e deduzir $P(k)$, mas devemos supor $k-1 \geq n_0$.

Funcionamento:

- Provamos o caso base $P(n_0)$.
- Assumimos $P(k)$ para algum $k \geq n_0$.
- Deduzimos $P(k+1)$ (ou seja, provamos $P(k) \Rightarrow P(k+1)$).

Podemos usar

- incremento $m > 1$ no passo indutivo, mas devemos provar m casos base;
- $P(k-1)$ como hipótese e deduzir $P(k)$, mas devemos supor $k-1 \geq n_0$.

Tome muito cuidado para

- provar corretamente o caso base;
- não usar alguma hipótese falsa no passo indutivo (em particular, respeite o fato de que $k \geq n_0$).

Princípio da Indução Completa: continuação

Funcionamento:

- Provamos o caso base $P(n_0)$.
- Assumimos $P(n_0), P(n_0+1), ..., P(k)$ para algum $k \geq n_0$.
- Deduzimos $P(k+1)$ (ou seja, provamos

$$P(n_0) \wedge P(n_0+1) \wedge ... \wedge P(k) \Rightarrow P(k+1)$$

Princípio da Indução Completa (PIC)

Funcionamento:

- Provamos o caso base $P(n_0)$.
- Assumimos $P(n_0), P(n_0+1), ..., P(k)$ para algum $k \geq n_0$.
- Deduzimos $P(k+1)$ (ou seja, provamos

$$P(n_0) \land P(n_0+1) \land ... \land P(k) \Rightarrow P(k+1)$$

Ou seja,

$$\exists k \in \mathbb{N}, (k \geq n_0 \land (\forall i \in \mathbb{N}, n_0 \leq i \leq k \Rightarrow P(i)))$$

Funcionamento:

- Provamos o caso base $P(n_0)$.
- Assumimos $P(n_0), P(n_0+1), ..., P(k)$ para algum $k \geq n_0$.
- Deduzimos $P(k+1)$ (ou seja, provamos

$$P(n_0) \wedge P(n_0+1) \wedge ... \wedge P(k) \Rightarrow P(k+1)$$

Ou seja,

$$\exists k \in \mathbb{N}, (k \geq n_0 \wedge (\forall i \in \mathbb{N}, n_0 \leq i \leq k \Rightarrow P(i)))$$

Como no PIM, usamos a hipótese de indução para provar $P(k+1)$

Mais um exemplo de prova por PIC

- 0 e 1 não são primos por definição.
- 2 e 3 são primos
- $4 = 2 \cdot 2$, logo, um produto de primos
- 5 é primo
- $6 = 2 \cdot 3$, logo, um produto de primos
- ...
- $1237848 = 2^3 \cdot 3 \cdot 51577$, logo, um produto de primos
- ...

Mais um exemplo de prova por PIC

- 0 e 1 não são primos por definição.
- 2 e 3 são primos
- $4 = 2 \cdot 2$, logo, um produto de primos
- 5 é primo
- $6 = 2 \cdot 3$, logo, um produto de primos
- ...
- $1237848 = 2^3 \cdot 3 \cdot 51577$, logo, um produto de primos
- ...

Teorema

Todo natural $n \geq 2$ é primo ou um produto de números naturais primos.

Bônus: Unicidade dos fatores primos

Teorema

Além disso, esse produto é único (exceto pela ordem dos fatores).

Teorema

Além disso, esse produto é único (exceto pela ordem dos fatores).

Antes de provar a unicidade, precisamos do seguinte resultado:

Lema

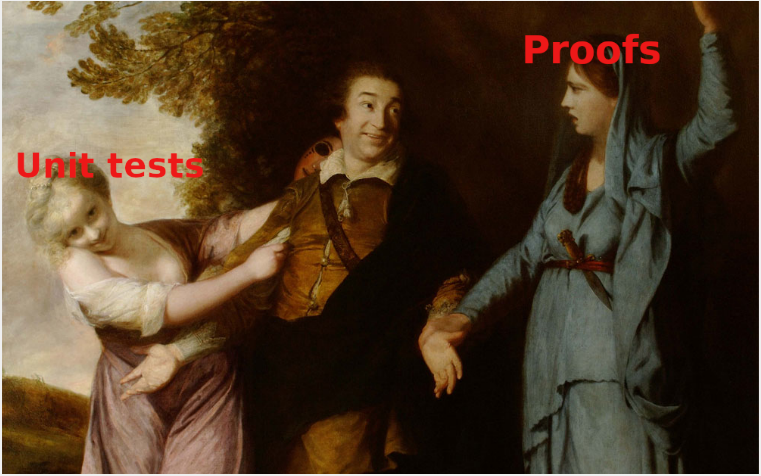
Se p é primo, então, para todo $a, b \in \mathbb{Z}$,

$$p \mid (a \cdot b) \Rightarrow (p \mid a \text{ ou } p \mid b)$$

Provando corretude de algoritmos

Como você sabe que seu código está correto?

Simplicidade versus garantia de corretude.



Provando que um algoritmo é correto

Temos que provar duas coisas:

1. O algoritmo termina?
2. Se termina, ele devolve a resposta esperada?

Provando a terminação de um algoritmo

- Se não há laço de repetição ou recursão, então o algoritmo termina.

Provando a terminação de um algoritmo

- Se não há laço de repetição ou recursão, então o algoritmo termina.
- Em muitos casos, há um laço com número de iterações claros (como $\mathtt{for}(\mathtt{i} = 0; \mathtt{i} < \mathtt{n}; \mathtt{i} + +)$) e a terminação é evidente.

Provando a terminação de um algoritmo

- Se não há laço de repetição ou recursão, então o algoritmo termina.
- Em muitos casos, há um laço com número de iterações claros (como $\texttt{for}(\texttt{i} = 0; \texttt{i} < \texttt{n}; \texttt{i}++)$) e a terminação é evidente.
- Em outros casos, é preciso identificar uma condição de parada para o laço e mostrar que ela é alcançada eventualmente.

Provando a terminação de um algoritmo

- Se não há laço de repetição ou recursão, então o algoritmo termina.
- Em muitos casos, há um laço com número de iterações claros (como $\texttt{for}(\texttt{i} = 0; \texttt{i} < \texttt{n}; \texttt{i} + +)$) e a terminação é evidente.
- Em outros casos, é preciso identificar uma condição de parada para o laço e mostrar que ela é alcançada eventualmente.
  - Por exemplo: $\texttt{while}(\texttt{abs}(\texttt{a} * \texttt{b}) > 0)$: poderíamos mostrar que o corpo do laço diminui o valor de `a` ou de `b` de tal forma que, em algum momento, ao menos um deles será igual a zero.

Basicamente, o mesmo que uma prova por indução matemática:

- Definimos $P(n)$ como alguma propriedade que garante a corretude do algoritmo para entradas de tamanho n"
- Provamos um caso base, como $P(0)$ ou $P(1)$.
- Provamos $P(k) \Rightarrow P(k+1)$

Provando que o algoritmo devolve a resposta esperada

Basicamente, o mesmo que uma prova por indução matemática:

- Definimos $P(n)$ como alguma propriedade que garante a corretude do algoritmo para entradas de tamanho n"
- Provamos um caso base, como $P(0)$ ou $P(1)$.
- Provamos $P(k) \Rightarrow P(k+1)$

Mas como estruturar essa prova:

- Algoritmo recursivo: a prova é mais natural
- Algoritmo iterativo: invariantes de laços

Exemplo: função `Min` recursiva

Algorithm: $\texttt{Min}(v, n)$

Input: vetor $v[0, ..., n-1] \in \mathbb{Z}^n$, $n \geq 1$

Output: mínimo de $\{v[0], v[1], ..., v[n-1]\}$

```
if n = 1 then
    return v[0]
else
    x = Min(v, n − 1)
    if v[n − 1] < x then
        x = v[n − 1]
    return x
```

Exemplo: função `Min` recursiva

Algorithm: $\texttt{Min}(v, n)$

Input: vetor $v[0, ..., n-1] \in \mathbb{Z}^n$, $n \geq 1$

Output: mínimo de $\{v[0], v[1], ..., v[n-1]\}$

```
if n = 1 then
    return v[0]
else
    x = Min(v, n − 1)
    if v[n − 1] < x then
        x = v[n − 1]
    return x
```

- Terminação: fácil de ver ($n-1$ chamadas recursivas).

Exemplo: função `Min` recursiva

Algorithm: `Min`(v, n)

Input: vetor $v[0, ..., n-1] \in \mathbb{Z}^n$, $n \geq 1$
Output: mínimo de $\{v[0], v[1], ..., v[n-1]\}$

```
if n = 1 then
    return v[0]
else
    x = Min(v, n − 1)
    if v[n − 1] < x then
        x = v[n − 1]
    return x
```

- Terminação: fácil de ver ($n-1$ chamadas recursivas).
- Caso base: Para $n = 1$, o algoritmo é trivialmente correto, pois se um elemento é o único do conjunto, então ele é o mínimo.

Exemplo: função `Min` recursiva

Algorithm: $\mathtt{Min}(v, n)$

Input: vetor $v[0, ..., n-1] \in \mathbb{Z}^n$, $n \geq 1$
Output: mínimo de $\{v[0], v[1], ..., v[n-1]\}$

```
if n = 1 then
    return v[0]
else
    x = Min(v, n − 1)
    if v[n − 1] < x then
        x = v[n − 1]
    return x
```

- Terminação: fácil de ver ($n-1$ chamadas recursivas).
- Caso base: Para $n = 1$, o algoritmo é trivialmente correto, pois se um elemento é o único do conjunto, então ele é o mínimo.
- Passo indutivo: Assumimos a corretude para $n - 1 \geq 1$ e provamos que é correto para n.

Exemplo: função `Min` iterativa

Algorithm: `Min`

Input: vetor $v[0, \ldots, n-1] \in \mathbb{Z}^n$, $n \geq 1$

Output: mínimo de $\{v[0], v[1], \ldots, v[n-1]\}$

$x = v[0]$

▷ `Esta linha marca o caso base`

for $2 \leq k \leq n-1$ **do**

  ▷ `Esta linha marca o início da iteração`

  if $v[k] < x$ **then**

    $x = v[k]$

  ▷ `Esta linha marca o final da iteração`

return x

Exemplo: função `Min` iterativa

Algorithm: `Min`

Input: vetor $v[0, \dots, n-1] \in \mathbb{Z}^n$, $n \geq 1$

Output: mínimo de $\{v[0], v[1], \dots, v[n-1]\}$

$x = v[0]$

▷ `Esta linha marca o caso base`

for $2 \leq k \leq n-1$ do

    ▷ `Esta linha marca o início da iteração`

    if $v[k] < x$ then

        $x = v[k]$

    ▷ `Esta linha marca o final da iteração`

return x

- Terminação: trivial.

Exemplo: função Min iterativa

Algorithm: Min

Input: vetor $v[0, ..., n-1] \in \mathbb{Z}^n$, $n \geq 1$

Output: mínimo de $\{v[0], v[1], ..., v[n-1]\}$

$x = v[0]$

▷ Esta linha marca o caso base

for $2 \leq k \leq n-1$ do

  ▷ Esta linha marca o início da iteração

  if $v[k] < x$ then

    $x = v[k]$

  ▷ Esta linha marca o final da iteração

return x

- Terminação: trivial.
- Caso base: É o que acontece antes de o laço ser executado (caso $n = 1$...).

Exemplo: função `Min` iterativa

Algorithm: `Min`

Input: vetor $v[0, ..., n-1] \in \mathbb{Z}^n$, $n \geq 1$
Output: mínimo de $\{v[0], v[1], ..., v[n-1]\}$

$x = v[0]$
▷ `Esta linha marca o caso base`
for $2 \leq k \leq n-1$ **do**
    ▷ `Esta linha marca o início da iteração`
    if $v[k] < x$ **then**
        $x = v[k]$
    ▷ `Esta linha marca o final da iteração`
return x

- Terminação: trivial.
- Caso base: É o que acontece antes de o laço ser executado (caso $n = 1$...).
- Invariante do laço: propriedade que é válida no início e no fim de cada iteração (passo indutivo...).

Exemplo: Divisão euclidiana

Algorithm: `DivEuc`

Input: $a, b \in \mathbb{N}$ com $b \neq 0$.

Output: $q, r \in \mathbb{Z}$ tais que $a = bq + r$ e $0 \leq r < b$

```
if a < b then
    return 0, a
q = 0
r = a
▷ Esta linha marca o caso base
while r ≥ b do
    ▷ Esta linha marca o início da iteração
    q = q + 1
    r = r − b
    ▷ Esta linha marca o final da iteração
return q, r
```

Exemplo: Divisão euclidiana

```
Algorithm: DivEuc
Input:  a, b ∈ ℕ com b ≠ 0.
Output: q, r ∈ ℤ tais que a = bq + r e 0 ≤ r < b
if a < b then
    return 0, a
q = 0
r = a
▷ Esta linha marca o caso base
while r ≥ b do
    ▷ Esta linha marca o início da iteração
    q = q + 1
    r = r − b
    ▷ Esta linha marca o final da iteração
return q, r
```

- Terminação: ??

Exemplo: Divisão euclidiana

```
Algorithm: DivEuc
Input:  a, b ∈ ℕ com b ≠ 0.
Output: q, r ∈ ℤ tais que a = bq + r e 0 ≤ r < b
if a < b then
    return 0, a
q = 0
r = a
▷ Esta linha marca o caso base
while r ≥ b do
    ▷ Esta linha marca o início da iteração
    q = q + 1
    r = r − b
    ▷ Esta linha marca o final da iteração
return q, r
```

- Terminação: ??
- Caso base: O que podemos concluir sobre $a, b, q,$ e r?

Exemplo: Divisão euclidiana

Algorithm: DivEuc

Input: $a, b \in \mathbb{N}$ com $b \neq 0$.
Output: $q, r \in \mathbb{Z}$ tais que $a = bq + r$ e $0 \leq r < b$

```
if a < b then
    return 0, a
q = 0
r = a
▷ Esta linha marca o caso base
while r ≥ b do
    ▷ Esta linha marca o início da iteração
    q = q + 1
    r = r − b
    ▷ Esta linha marca o final da iteração
return q, r
```

- Terminação: ??
- Caso base: O que podemos concluir sobre a, b, q, e r?
- Invariante:

Exemplo: Divisão euclidiana

Algorithm: DivEuc

Input: $a, b \in \mathbb{N}$ com $b \neq 0$.
Output: $q, r \in \mathbb{Z}$ tais que $a = bq + r$ e $0 \leq r < b$

```
if a < b then
    return 0, a
q = 0
r = a
▷ Esta linha marca o caso base
while r ≥ b do
    ▷ Esta linha marca o início da iteração
    q = q + 1
    r = r − b
    ▷ Esta linha marca o final da iteração
return q, r
```

- Terminação: ??
- Caso base: O que podemos concluir sobre a, b, q, e r?
- Invariante:
  - Assumimos que $a = bq + r$ no início da iteração e mostramos que $a = bq + r$ ainda vale no final.

Exemplo: Divisão euclidiana

```
Algorithm: DivEuc
Input:  a, b ∈ ℕ com b ≠ 0.
Output: q, r ∈ ℤ tais que a = bq + r e 0 ≤ r < b
if a < b then
    return 0, a
q = 0
r = a
▷ Esta linha marca o caso base
while  r ≥ b do
    ▷ Esta linha marca o início da iteração
    q = q + 1
    r = r − b
    ▷ Esta linha marca o final da iteração
return q, r
```

- Terminação: ??
- Caso base: O que podemos concluir sobre a, b, q, e r?
- Invariante:
  - Assumimos que $a = bq + r$ no início da iteração e mostramos que $a = bq + r$ ainda vale no final.
  - Além disso, como mostrar que $0 \leq r < b$ no *return*? (Note que sair do laço implica $r \geq b$ ser falso, logo $r < b$. Além disso, no início da iteração, temos $r \geq b$. No corpo do laço, subtraímos b de r, então no final da iteração temos $r \geq 0$. Portanto, quando saímos do laço, temos $0 \leq r < b$).

Perguntas, observações, comentários?