Projeto e Análise de Algoritmos (2025-1)

Flávio L. C. de Moura*

27 de março de 2025

A correção de algoritmos (continuação)

Dados um natural x e uma lista l, queremos responder se x ocorre ou não em l. Um algoritmo recursivo que resolve este problema retornando `true` se o número estiver na lista, e `false` caso contrário, é dado a seguir:

$$in\_list\ x\ l = \begin{cases} false, & \text{se } l = nil \\ true, & \text{se } l = h :: l' \text{ e } x = h \\ in\_list\ x\ l', & \text{se } l = h :: l' \text{ e } x \neq h \end{cases}$$

Afirmação. Qualquer que seja a lista l e o natural x, se x ocorre em l então $in\_list\ l = true$ e se x não ocorre em l então $in\_list\ l = false$.

Prova da afirmação. A prova é por indução na estrutura da lista l.

1. Se l for lista vazia então $in\_list\ x\ nil = false$ e a afirmação é verdadeira uma vez que x não ocorre na lista vazia.

2. Suponha que l tem a forma $h :: tl$. Temos 2 subcasos:

   (a) $x = h$: Neste caso, encontramos uma ocorrência de x e o algoritmo retorna `true` como esperado.

   (b) $x \neq h$: Neste caso, o algoritmo continua a busca na cauda da lista l, e a hipótese de indução nos permite concluir que a afirmação está correta.

A prova que acabamos de fazer é bem simples, mas veremos outras bem mais complicadas ao longo do semestre. A prova acima é dita informal em contraposição com provas mecânicas, isto é, feitas em sistema de provas immplementado em um computador. Estes sistemas de provas são chamados de assistentes de provas e existem vários disponíveis, como o Coq, PVS e Isabele/HOL.

*flaviomoura@unb.br

Aqui veremos como refazer a prova acima no Coq.

```
Fixpoint in_list x l := match l with
                        | nil => false
                        | h::tl => if x =? h then true else (in_list x tl)
                        end.

Lemma in_list_true: forall l x, In x l <-> in_list x l = true.
```

Agora considere o problema de encontrar o menor elemento de uma lista de números naturais. A função recursiva $min\_list\ h\ l$ que recebe um natural h e uma lista de números naturais l como argumento, e retorna o menor elemento da lista $(h :: l)$:

$$min\_list\ h\ l = \begin{cases} h, & \text{se } l = nil \\ min\_list\ h\ l', & \text{se } l = h' :: l' \text{ e } h \leq h' \\ min\_list\ (h', l'), & \text{se } l = h' :: l' \text{ e } h > h' \end{cases}$$

A função é definida recursivamente na estrutura da lista l, segundo argumento da função min_list. De fato, quando l é a lista vazia, notação nil, a função retorna h. Quando a lista é não vazia com primeiro elemento h' e cauda l', notação $h' :: l'$, temos dois subcasos a considerar:

1. $h \leq h'$: Neste caso, a chamada recursiva é feita com os argumentos h e l';

2. $h > h'$: Neste caso, a chamada recursiva é feita com os argumentos h' e l'.

Afirmação. A função $min\_list\ h\ l$ retorna o menor elemento da lista $(h :: l)$.

Podemos ver facilmente que a propriedade expressa nesta afirmação é verdadeira em diversos casos. Por exemplo, se $h = 1$ e $l = 2 :: 3 :: 4 :: nil$ então esperamos que $min\_list1(2 :: 3 :: 4 :: nil)$ retorne 1, ou seja, o menor elemento da lista $(1 :: 2 :: 3 :: 4 :: nil)$. De fato, temos

$$\begin{aligned} &min\_list\ 1\ (2 :: 3 :: 4 :: nil) &&= \\ &min\_list\ 1\ (3 :: 4 :: nil) &&= \\ &min\_list\ 1\ (4 :: nil) &&= \\ &min\_list\ 1\ nil &&= \ 1 \end{aligned}$$

Ou ainda,

$$\begin{aligned} &min\_list\ 10\ (2 :: 3 :: 4 :: nil) &&= \\ &min\_list\ 2\ (3 :: 4 :: nil) &&= \\ &min\_list\ 2\ (4 :: nil) &&= \\ &min\_list\ 2\ nil &&= \ 2 \end{aligned}$$

E assim, podemos testar diversos argumentos para verificar se a função (algoritmo) min_list está funcionando corretamente, mas esses testes seriam suficiente para garantirmos que a afirmação acima é verdadeira? Observe que existe uma infinidade de argumentos distintos possíveis... Certamente, não! Para garantirmos que função $min\_list\ h\ l$ retorna o menor elemento da lista $(h :: l)$

quaisquer que sejam h e l precisamos construir uma prova. Utilizando indução na estrutura da lista l, provaremos que a afirmação é verdadeira.

Prova da afirmação. Temos dois casos a considerar:

1. A lista l é vazia, isto é, $l = nil$: Neste caso, $min\_list\ h\ nil$ retorna h, que é o menor elemento da lista $(h :: nil)$.

2. A lista $l = h' :: l'$, isto é, l tem primeiro elemento h' e cauda l': Aqui temos que considerar duas situações de acordo com a definição de min_list:

   (a) $h \leq h'$: Neste caso, temos por hipótese de indução que $min\_list\ h\ l'$ retorna o menor elemento da lista $(h :: l')$, e como $h \leq h'$, podemos dizer que h' não é o menor elemento da lista $(h :: h' :: l')$. E portanto, o menor elemento da lista $(h :: h' :: l')$ é também o menor elemento da lista $(h :: l')$ que é retornado pela função min_list pela hipótese de indução. Isto mostra que a afirmação é verdadeira neste caso.

   (b) $h > h'$: Neste caso, o menor elemento da lista $(h :: h' :: l')$ é igual ao menor elemento da lista $(h' :: l')$, e por hipótese de indução este é o elemento retornado pela chamada recursiva $min\_list\ h'\ l'$, o que finaliza a prova de que a afirmação é verdadeira. □

A definição da função e o enunciado da propriedade que corresponde a afirmação que acabamos de provar são dadas a seguir no assistente de provas Coq:

```
Fixpoint min_list h l := match l with
                         | nil => h
                         | h'::l' => if h <=? h'
                                     then min_list h l'
                                     else min_list h' l'
                         end.

Definition le_all h l := forall x, In x l -> h <= x.


Lemma min_list_minimum: forall l h, le_all (min_list h l) (h::l).
```

Uma versão não recursiva do algoritmo min_list

Estudaremos diversos algoritmos não-recursivos neste curso, e a seguir, apresentamos uma versão não-recursiva do algoritmo min_list. Neste caso, o algoritmo recebe o vetor $A[1..n]$ como argumento:

Algorithm 1: $min\_list\_norec(A[1..n])$

```
1 min ← A[1];
2 for i = 2 to n do
3 |   if A[i] < min then
4 |   |   min ← A[i];
5 |   end
6 end
7 return min;
```

Afirmação. Ao final de sua execução o algoritmo $min\_list\_norec(A[1..n])$ retorna o menor elemento do vetor $A[1..n]$.

Como provar esta afirmação? Em programas contendo laços utilizamos as chamadas *invariantes de laço*, que são propriedades satisfeitas durante toda a execução de um laço. A prova de uma invariante de laço consiste em três etapas:

1. **Inicialização**: Nesta etapa mostramos que a propriedade é satisfeita antes da primeira execução do laço. Esta etapa é equivalente à base da indução em uma prova indutiva;

2. **Manutenção**: Esta é a etapa mais delicada da prova (veja a semelhança com um passo indutivo). Aqui assumimos por hipótese (de indução) que a invariante vale antes de uma iteração arbitrária do laço (depois da primeira) e mostramos que a invariante continua válida antes da próxima iteração. Ou seja, assumimos que a invariante vale antes da k-ésima iteração ($k > 0$) e mostramos que ao final desta iteração, isto é, antes da $(k+1)$-ésima iteração, a invariante continua valendo;

3. **Finalização**: Nesta etapa concluímos que a invariante é satisfeita durante toda a execução do laço, e esta informação é utilizada para estabelecer a correção do algoritmo.

O laço **for** do algoritmo $min\_list\_norec(A[1..n])$ percorre os todos os elementos do vetor A a partir da segunda posição. Não podemos garantir que a variável min armazena o menor elemento de A ao longo de toda a execução do algoritmo, mas certamente min armazena o menor elemento dos que já foram considerados. Assim, provaremos a seguinte invariante de laço:

Invariante. Antes de cada iteração indexada por i, a variável min contém o menor elemento do subvetor $A[1..i-1]$.

Prova da invariante. Consideraremos as três etapas descritas acima:

1. **Inicialização**: Antes da primeira iteração, temos $i = 2$, e $min = A[1]$ (linha 1) contém o menor elemento do subvetor $A[1]$.

2. **Manutenção**: Assuma que antes da k-ésima iteração, isto é, quando $i = k + 1$, *min* contém o menor elemento do subvetor $A[1..k]$. Ao longo da k-ésima iteração o elemento $A[i] = A[k + 1]$ será comparado com *min* (linha 3). Se $A[k + 1]$ for menor do que *min* então encontramos um valor menor do que o atual no subvetor $A[1..k + 1]$, e este novo valor é armazenado em *min* (linha 4). Assim, antes da (k+1)-ésima iteração *min* contém o menor elemento do subvetor $A[1..k + 1]$, como queríamos provar.

3. **Finalização**: Ao final da execução do laço, isto é, quando $i = n + 1$, temos que *min* contém o menor elemento do vetor $A[1..n]$, e portanto a invariante é verdadeira. □

Exercícios

1. Apresente o pseudocódigo de um algoritmo não-recursivo que verifica se um determinado número está presente em um vetor $A[1..n]$ de números naturais. O algoritmo deve retornar `true` se o número estiver na lista e `false` caso contrário. Em seguida, prove a correção do seu algoritmo.

2. Apresente o pseudocódigo de um algoritmo recursivo que retorna o número de ocorrências de um dado elemento em uma lista. Em seguida, prove a correção do seu algoritmo.

3. Apresente o pseudocódigo de um algoritmo não-recursivo que retorna o número de ocorrências de um dado elemento em um vetor. Em seguida, prove a correção do seu algoritmo.

4. Apresente o pseudocódigo de um algoritmo recursivo que retorna o maior elemento de uma lista de números naturais. Em seguida, prove a correção do seu algoritmo.

5. Apresente o pseudocódigo de um algoritmo não-recursivo que retorna o maior elemento de um vetor $A[1..n]$ de números naturais. Em seguida, prove a correção do seu algoritmo.

6. Apresente o pseudocódigo de uma versão recursiva do algoritmo *selection sort* que ordena uma lista de números naturais via a seleção do maior elemento. Em seguida, prove a correção do seu algoritmo.

7. Apresente o pseudocódigo de uma versão não-recursiva do algoritmo *selection sort* que ordena uma lista de números naturais via a seleção do maior elemento. Em seguida, prove a correção do seu algoritmo.

8. Apresente o pseudocódigo de uma versão recursiva do algoritmo *selection sort* que ordena uma lista de números naturais via a seleção do menor elemento. Em seguida, prove a correção do

seu algoritmo.

9. Apresente o pseudocódigo de uma versão não-recursiva do algoritmo *selection sort* que ordena uma lista de números naturais via a seleção do menor elemento. Em seguida, prove a correção do seu algoritmo.