Ordenação por inserção (parte 1)

Projeto e Análise de Algoritmos (2023-2)

Flávio L. C. de Moura*

Nesta seção estudaremos o algoritmo de ordenação por inserção recursivo. A estrutura de dados utilizada é a de listas, e para simplificar trabalharemos com números naturais, mas as ideias são as mesmas para ordenarmos qualquer estrutura que possua uma ordem total. Vimos no capítulo anterior que as listas de naturais possuem dois construtores: nil para representar a lista vazia, e o operador :: que nos permite construir uma nova lista a partir de um número natural e de uma lista. Assim, a lista unitária contendo apenas o natural 5 é representada por $5 :: nil$, enquanto que a lista $1 :: (5 :: nil)$, ou simplesmente $1 :: 5 :: nil$, representa a lista que tem 1 como primeiro elemento, e a lista $5 :: nil$ como cauda.

A operação que dá nome ao algoritmo é a operação de inserção porque a cada passo queremos inserir um novo elemento em uma lista já ordenada. Suponha, por exemplo, que queiramos inserir o número 3 na lista $1 :: 5 :: nil$, isto é, o nosso objetivo final é obter a lista ordenada $1 :: 3 :: 5 :: nil$. Para isto, precisamos inicialmente comparar o 3 com o primeiro elemento da lista, e o resultado desta comparação nos diz que o 3 deve ser inserido depois do 1, ou seja, em algum lugar da cauda da lista. Em seguida, comparamos o 3 com o primeiro elemento da cauda, ou seja, com 5, e como $3 < 5$, sabemos que ele deve ser inserido antes do 5. Esta ideia está implementada na função $insere$ definida a seguir:

Definição 1. *Sejam x um número natural, e l uma lista de números naturais. A função $(insere\ x\ l)$ que insere o natural x na lista l é definida recursivamente como a seguir:*

$$insere\ x\ l = \begin{cases} x :: nil, & se\ l = nil \\ x :: l, & se\ l = h :: tl\ e\ x \leq h \\ h :: (insere\ x\ tl), & se\ l = h :: tl\ e\ x > h \end{cases}$$

O algoritmo de ordenação por inserção então consiste em recursivamente, dada uma lista não vazia $h :: tl$, inserir o primeiro elemento h na versão ordenada da cauda tl. Ou seja, o algoritmo de ordenação por inserção que será implementado pela função de $ord\_insercao$ vai receber como argumento uma lista l para ordenar. Se l for a lista vazia não há nada a fazer, e caso contrário, recursivamente ordenamos a cauda da lista para então inserir o novo elemento:

Definição 2. *Seja l uma lista de números naturais. A função $ord\_insercao$ é definida recursivamente como a seguir:*

$$ord\_insercao\ l = \begin{cases} nil, & se\ l = nil \\ insere\ h\ (ord\_insercao\ tl), & se\ l = h :: tl \end{cases}$$

Vejamos como este algoritmo funciona na prática. Suponha que queiramos ordenar a lista $3 :: 2 :: 1 :: nil$. Ao fornecermos esta lista como argumento para a função $ord\_insercao$, temos:

*flaviomoura@unb.br

$ord\_insercao\ (3 :: 2 :: 1 :: nil) =$ (def. $ord\_insercao$)
$insere\ 3\ (ord\_insercao\ (2 :: 1 :: nil)) =$ (def. $ord\_insercao$)
$insere\ 3\ (insere\ 2\ (ord\_insercao\ (1 :: nil))) =$ (def. $ord\_insercao$)
$insere\ 3\ (insere\ 2\ (insere\ 1\ (ord\_insercao\ nil))) =$ (def. $ord\_insercao$)
$insere\ 3\ (insere\ 2\ (insere\ 1\ nil)) =$ (def. $insere$)
$insere\ 3\ (insere\ 2\ (1 :: nil)) =$ (def. $insere$)
$insere\ 3\ (1 :: (insere\ 2\ nil)) =$ (def. $insere$)
$insere\ 3\ (1 :: 2 :: nil) =$ (def. $insere$)
$1 :: (insere\ 3\ (2 :: nil)) =$ (def. $insere$)
$1 :: 2 :: (insere\ 3\ nil) =$ (def. $insere$)
$1 :: 2 :: 3 :: nil$

Veja que o algoritmo ordenou corretamente a lista $3 :: 2 :: 1 :: nil$, mas será que ele ordena corretamente qualquer lista de números naturais? Para responder esta pergunta, vamos analisar se o algoritmo é correto ou não. Para isto precisaremos definir algumas noções que serão utilizadas também em outros algoritmos de ordenação. A primeira noção que precisamos definir formalmente é a de ordenação. Ou seja, o que significa dizer que uma lista está ordenada? A definição a seguir apresenta o predicado *sorted* que caracteriza a noção de lista ordenada:

Definição 3. *Sejam x e y números naturais, e l uma lista de números naturais. O predicado sorted, que caracteriza o fato de uma lista estar ordenada, é definido por meio das seguintes regras de inferência:*

$$\frac{}{sorted\ nil}\ (sorted\_nil) \qquad \frac{}{sorted\ x :: nil}\ (sorted\_one)$$

$$\frac{x \leq y \qquad sorted\ y :: l}{sorted\ x :: y :: l}\ (sorted\_all)$$

A regra $(sorted\_nil)$ é um axioma que estabelece que a lista vazia está ordenada. A regra $(sorted\_one)$ também é um axioma que estabelece que listas unitárias estão ordenadas. A regra $(sorted\_all)$ possui duas condições para que uma lista da forma $x :: y :: l$ esteja ordenada: $x \leq y$ e a lista $y :: l$ tem que estar ordenada. Em outras palavras, a regra $(sorted\_all)$ diz que uma lista com pelo menos dois elementos está ordenada, se o primeiro elemento é menor ou igual ao segundo elemento, e a cauda da lista (ou seja, a lista do segundo elemento em diante) está ordenada. Note que as variáveis x, y e a lista l estão implicitamente quantificadas universalmente na Definição 3. Segundo esta definição, a lista $(1 :: 2 :: 3 :: nil)$ está ordenada. De fato, a prova deste fato é dada pela seguinte árvore de derivação:

$$\dfrac{1 \leq 2 \qquad \dfrac{2 \leq 3 \qquad \dfrac{}{sorted\ (3 :: nil)}\ (sorted\_one)}{sorted\ (2 :: 3 :: nil)}\ (sorted\_all)}{sorted\ (1 :: 2 :: 3 :: nil)}\ (sorted\_all)$$

Com esta definição podemos provar a primeira parte da correção do algoritmo $ord\_insercao$:

Teorema 4. *A lista* $(ord\_insercao\ l)$ *é ordenada, qualquer que seja a lista* l*. Em outras palavras, temos que sorted* $(ord\_insercao\ l)$*.*

A prova deste teorema foi feito na aula de hoje assumindo o seguinte lema:

Lema 5. *Sejam* x *um número natural, e* l *uma lista de números naturais quaisquer. Se* l *está ordenada então* $(insere\ x\ l)$ *também está ordenada.*

1 Exercícios propostos

1. Prove do Teorema 4. A estratégia de prova é indução na estrutura da lista l.[1]
2. Prove o Lema 5.
3. Mostre que o Teorema 4 estabelece apenas a correção parcial do algoritmo *ord_insercao*. Ou seja, mostre que existem algoritmos de ordenação incorretos, mas que satisfazem o Teorema 4.

[1] Esta prova foi feita na aula passada, mas esta é uma boa oportunidade de escrever a prova sozinho e verificar se os conceitos sobre indução estão claros.