Algoritmos de Ordenação em LISP

Léo Willian Kölln

8 de Agosto de 2006

Curso de Ciências da Computação
Programação Funcional - INE5363

INE - Departamento de Informática e Estatística
CTC - Centro Tecnológico
UFSC - Universidade Federal de Santa Catarina

1 Bubble Sort

O Bubble Sort, ou ordenação por flutuação (literalmente "por bolha"), é um algoritmo de ordenação dos mais simples. A idéia é percorrer o vetor diversas vezes, a cada passagem fazendo flutuar para o topo o menor elemento da seqüência. Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

A complexidade desse algoritmo é de Ordem quadrática (n^2). Por isso, ele não é recomendado para programas que precisem de velocidade e operem com quantidade elevada de dados.

O algoritmo pode ser descrito em pseudocódigo como segue abaixo. A é um Vetor de elementos que podem ser comparados e n é o tamanho desse vetor.

1.1 Pseudocódigo

```
function bubblesort (A : list[1..n]) {
var int i, j;
for i from n downto 1 {
for j from 1 to i-1 {
if (A[j] > A[j+1])
swap(A[j], A[j+1])
}
}
}
```

1.2 Implementação do Bubble Sort em LISP

```
; Função para Trocar dados de posição em um Array
(defun LKSwap (lista i j)
(setq tmp (aref lista i))
(setf (aref lista i) (aref lista j))
(setf (aref lista j) tmp)
)

; Implementação do Bubble Sort em LISP
(defun LKBubbleSort (lista)
(do ((i (length lista) (- i 1))) ((= i 0))
(do ((j 0 (+ j 1))) ((= j (- i 1)))
(if (> (aref lista j) (aref lista (+ j 1)))
(LKSwap lista j (+ j 1))
)
)
)
)
```

2 Merge Sort

O Merge Sort, ou ordenação por mistura, é um exemplo de algoritmo de ordenação do tipo dividir-para-conquistar.

Sua idéia básica é que é muito fácil criar uma seqüência ordenada a partir de duas outras também ordenadas. Para isso, ele divide a seqüência original em pares de dados, ordena-as; depois as agrupa em seqüências de quatro elementos, e assim por diante, até ter toda a seqüência dividida em apenas duas partes.

Os três passos úteis dos algoritmos dividir-para-conquistar, ou "divide and conquer", que se aplicam ao merge sort são:

1. Dividir: Dividir os dados em subseqüências pequenas
2. Conquistar: Classificar as duas metades recursivamente aplicando o merge sort
3. Combinar: Juntar as duas metades em um único conjunto já classificado

2.1 Pseudocódigo

```
function mergesort(m)
var list left, right
if length(m) = 1
return m
else
middle = length(m) / 2
for each x in m up to middle
add x to left
for each x in m after middle
add x to right
left = mergesort(left)
right = mergesort(right)
result = merge(left, right)
return result
```

2.2 Implementação do Merge Sort em LISP

```
; Função recursiva do Algoritmo
(defun mergeSort (lista)
(if (minimo lista) lista
(juntaListas
(mergeSort (MetadeEsquerda lista)) ; Metade da esquerda
(mergeSort (metadeDireita lista)) ; Metade da direita
)
)
)

; Define de a lista já é mínima ou não
(defun minimo (lista)
```

```
(or
(eq (length lista) 0)
(eq (length lista) 1)
)
)

; Função para a metade Direita
(defun metadeDireita (lista)
(last lista (ceiling (/ (length lista) 2)))
)

; Função para a metade Esquerda
(defun MetadeEsquerda (lista)
(ldiff lista (metadeDireita lista))
)

; Função para junção das Listas
(defun juntaListas (lista1 lista2)
(merge 'list lista1 lista2 #'<)
)
```

3 Quick Sort

O algoritmo Quick Sort é um método de ordenação muito rápido e eficiente, inventado por C.A.R. Hoare em 1960, quando visitou a Universidade de Moscou como estudante. Foi publicado em 1962 após uma série de refinamentos.

Sua premissa básica é dividir o problema de ordenar n itens em dois problemas menores. Em seguida, os problemas menores são ordenados de forma independente e os resultados são combinados para produzir a solução do problema todo.

3.1 Pseudocódigo

```
function quicksort(q)
var list less, pivotList, greater
if length(q) = 1
return q
select a pivot value pivot from q
for each x in q except the pivot element
if x < pivot then add x to less
if x = pivot then add x to greater
add pivot to pivotList
return concatenate(quicksort(less), pivotList, quicksort(greater))
```

3.2 Implementação do Quick Sort em LISP

```
; Função de Quick-Sort
(defun quickSort (lista)
(let*
```

```
(
(listaDupla nil)
(esquerda nil)
(direita nil)
(esquerdaOrdenada nil)
(direitaOrdenada nil)
)
(cond
((null lista) nil)
(t
(setq listaDupla (partition (car lista) (cdr lista)))
(setq esquerda (car listaDupla))
(setq direita (cdr listaDupla))
(setq esquerdaOrdenada (qs esquerda))
(setq direitaOrdenada (qs direita))
(append esquerdaOrdenada
(cons (car lista) direitaOrdenada)
)
)
)
)
)
```

Referências

[1] Wikipedia, **Bubble Sort** - http://en.wikipedia.org/wiki/Bubble_sort

[2] Wikipedia, **Merge Sort** - http://en.wikipedia.org/wiki/Merge_sort

[3] Wikipedia, **Quick Sort** - http://en.wikipedia.org/wiki/Quicksort

[4] Jason Harrison, **Sorting Algorithms** - http://www.cs.ubc.ca/spider/harrison/Java/sorting-demo.html

[5] Paulo Feofiloff, **Projeto de Algoritmos** - http://www.ime.usp.br/ pf/algoritmos/aulas/quick.html

[6] **Sorting Algorithms** - http://c2.com/cgi/wiki?SortingAlgorithms

[7] NIST, **Sort** - http://www.nist.gov/dads/HTML/sort.html

[8] Eugene Jitomirsky, **The QuickSort Algorithm** - http://www.mycsresource.net/articles/programming/sorting_algos/quicksort