Aula Prática: Ordenação

Ordenação

Para que os elementos de uma lista sejam dispostos numa ordem predefinida, temos:

Dada uma lista de n elementos:

$[e_1, e_2, ..., e_n]$

O problema de ordenação dos elementos é uma permutação dos mesmos para alguma relação de ordem, por exemplo, ordem crescente:

$[e'_1, e'_2, ..., e'_n]$ tal que $e'_1 \leq e'_2 \leq \ldots \leq e'_n$

Um algoritmo de ordenação dispõe os elementos de uma dada lista numa determinada ordem. Este processo é muitas vezes realizado para que os elementos sejam obtidos de forma mais eficiente.

1) Função para Inserir numa lista ordenada

Suponha uma lista ordenada de elementos (ordem crescente). Quando desejamos inserir um novo elemento, devemos garantir que o mesmo "entre" na posição correta. Para isso, podemos usar a função ***insereOrdenado*** abaixo:

```
insereOrdenado :: Ord a => a -> [a] -> [a]
insereOrdenado e [] = [e]
insereOrdenado e (x:xs)
    | e<=x = e : x : xs
    | otherwise = x : insereOrdenado e xs
```

Podemos ter como exemplo os testes abaixo:

```
> insereOrdenado 2 [-2,0,1,4,5]
[-2,0,1,2,4,5]

> insereOrdenado 'f' "abcdefghhij"
"abcdeffghhij"

> insereOrdenado ("Marcos",13) [("Ana",56),("Jose",26),("Lucas",34),
("Pedro",10),("Roberto",22)]
[("Ana",56),("Jose",26),("Lucas",34),("Marcos",13),("Pedro",10),
("Roberto",22)]
```

Se a lista é vazia, o novo elemento se torna o único da lista. Caso não seja vazia, e ***x*** seja seu primeiro elemento, então a posição de inserção depende se ***e*** é menor que ***x***. Caso positivo, ***e*** é colocado antes de ***x***, e em caso negativo, ***e*** deve ser inserido em alguma posição do resto da lista.

2) Função para ordenar uma lista qualquer

Um método bastante conhecido de ordenação é denominado ***quicksort*** (ordenação rápida). Este método consiste em resolver o problema da ordenação por partes, através da escolha de um elemento arbitrário ***x***, chamado ***pivô***. Por exemplo, podemos escolher como elemento pivô o primeiro da lista e então posicioná-lo corretamente na lista resultante ordenada. Para isso, podemos criar a nova lista concatenando os menores que ***x***, seguidos de ***x***, seguidos dos elementos maiores que ***x***, como mostra a função ***quicksort*** abaixo:

```
quicksort :: Ord a -> [a ] -> [a ]
quicksort [ ] = [ ]
quicksort (x : xs) = quicksort menores ++ [x] ++ quicksort maiores
         where
           menores = [a | a <- xs, a <= x ]
           maiores = [b | b <- xs, b > x ]
```

Observando a função, verificamos que a lista vazia já está ordenada, e que uma lista não vazia pode ser ordenada através de um mecanismo de recursão múltipla, no qual a função ***quicksort*** é aplicada mais de uma vez na sua definição: uma vez para ordenar os menores que o primeiro, e outra vez para ordenar os maiores.

Exercícios:

1) Usando a função *quicksort* mostre os passos da ordenação para a chamada abaixo:

```
> quicksort [5,3,7,8,2,1,4]
```

2) O que acontece se o símbolo <= for substituído por < na definição da função *quicksort*? Faça o teste para a chamada abaixo:

```
> quicksort "mariana"
```

3) Altere a função ***quicksort*** de forma a obter os maiores e menores utilizando a função genérica ***filter***.

4) Outra função de ordenação, chamada ***mergesort*** (ordenação por mistura) é implementada através da criação de uma sequência ordenada a partir de duas outras também ordenadas. Seja a função mistura:

```
mistura [] y = y
mistura x [] = x
mistura (x:xs) (y:ys)
  | x <= y = x: mistura xs (y:ys)
  | otherwise = y: mistura (x:xs) ys
```

a) Analise a função mistura e teste-a para as entradas abaixo fornecendo os resultados:

```
> mistura [1,3,4,8] [2,6,7]

> mistura "abc" "ABc"
```

b) Implemente a função ***mergesort*** observando as regras:

1. Dividir: Dividir os dados em subsequências pequenas;
2. Classificar: Classificar as duas metades recursivamente aplicando a função ***mergesort*** (recursão múltipla) ;
3. Misturar: Juntar as duas metades em um único conjunto já classificado.