Armando Matos, 2008-2009

Tópicos Avançados em Algoritmos - Folha prática de exercícios 02

1. Merge
O algoritmo "merge" tem como dados 2 vectores ordenados `u[0..a-1]` e `v[0..b-1]` e determina eficientemente um vector `w[0..a+b-1]` constituído pelos elementos de `u` e `v`. Por exemplo

```
merge([2,5,5,5,9],[1,2,5,8]) --> [1,2,2,5,5,5,5,8,9]
```

   (a) Descreva numa linguagem informal (pseudo-código) o algoritmo "merge".

   (b) Mostre que o número máximo de comparações *envolvendo elementos dos vectores* é $a+b-1$ (o algoritmo que descreveu na alínea anterior deve ser tal que isso aconteca!).

2. Mergesort
(problema repetido da folha prática 01) O "mergesort" é um método de ordenação muito eficiente, mesmo no pior caso. Descrição, supondo que n é uma potência de 2:

   `mergesort(v[0...n-1],n]:`

   (a) Se $n = 1$: nada se faz

   (b) Se $n \geq 2$:
      – $\texttt{mergesort}(\texttt{v}[0 \cdots \texttt{n}/2-1], \texttt{n}/2)$ // ordena $\texttt{v}[0 \cdots \texttt{n}/2-1]$
      – $\texttt{mergesort}(\texttt{v}[\texttt{n}/2 \cdots \texttt{n}-1], \texttt{n}/2)$ // ordena $\texttt{v}[\texttt{n}/2 \cdots \texttt{n}-1]$
      – $\texttt{merge}(\texttt{v}[0 \cdots \texttt{n}/2-1], \texttt{v}[\texttt{n}/2 \cdots \texttt{n}-1]) \longrightarrow \texttt{v}[0 \cdots \texttt{n}-1]$

   (a) Ilustre a execução do mergesort para o vector

$$v[\,] = \boxed{9 \mid 7 \mid 8 \mid 5 \mid 1 \mid 3 \mid 6 \mid 2}$$

   (b) Indique uma recorrência para um majorante do número de comparações efectuadas (são todas efectuadas durante a execução dos `merge`) pelo algoritmo.

   (c) Resolva a recorrência pelo método "tabelar / suspeitar / demonstrar".

3. Pesquisa binária – correcção e análise
Este exercício trata do algoritmo da pesquisa binária. Pretende-se implementar, mostrar a correcção e analisar a eficiência do algoritmo. Ao contrário do que poderá à partida parecer, o desenho de uma versão correcta do algoritmo poderá não ser trivial. Existem livros de programação em que este algoritmo está errado, (ver alínea 3d); por outro lado, o uso de linguagens "evoluídas" (por exemplo "object oriented") parece não ajudar muito...

   (a) Implemente a pesquisa binária numa linguagem informal (pseudo-código).
      – A função deve chamar-se `pesqbin` e tem 4 argumentos: `pesqbin(x,v,a,b)`, sendo `x` o valor a procurar no vector `v`, entre os índices `a` e `b` (extremos incluídos).
      – Admite-se que todos os elementos do vector `v` são distintos e estão ordenados por ordem crescente.
      – Com `v[i..j]` representamos o sub-vector correspondente aos índices `i`, `i+1`, `i+2`,..., `j`. Quando `i` > `j` esse sub-vector é (naturalmente!) vazio.
      – A chamada inicial é da forma `pesqbin(x,v,1,n)`; admite-se que o primeiro índice de `v` é 1.
      – O resultado da chamada `pesqbin(x,v,a,b)` deve corresponder a uma computação que termina e com o seguinte resultado:
         – Se for $\texttt{x} = \texttt{v}[\texttt{i}]$ para algum `i` com $a \leq \texttt{i} \leq \texttt{b}$ (esse `i` é necessariamente único): o resultado é `i`
         – Caso contrário (`x` não está em `v[a..b]`): resultado -1.

(b) Mostre a correcção do algoritmo que implementou. É muitas vezes conveniente mostrar a correcção em 2 fases:

1. Correcção parcial: se a computação terminar, o resultado está correcto.
2. Terminação: o algoritmo termina sempre.

Nota. Pode separar os casos: (i) `x` ocorre em `v[1..n]` e (ii) `x` não ocorre em `v[1..n]`.
Nota. A definição de um invariante de ciclo (um predicado que se demonstra ser sempre válido quando é efectuado o teste do ciclo ("`while`", suponhamos) pode ser importante para a demonstração.

(c) Análise a eficiência da pesquisa binária. Pretende-se contar o número máximo de comparações efectuadas pelo algoritmo que implementou; só se devem contabilizar as comparações *que envolvem o vector* `v`. Supõe-se que uma comparação pode dar 3 resultados, "menor", "igual" ou "maior"; assim, numa execução de

```
...
if x == v[i]:
   ...
else if x< v:
   ...
else:
   ...
```

há apenas uma comparação. Para facilitar, comece por admitir que $\mathtt{n}$ é da forma $\mathtt{n} = 2^{\mathtt{p}} - 1$ para algum inteiro $\mathtt{p} \geq 1$.

(d) Leia o artigo em `http://www.ncc.up.pt/~acm/aulas/aa/modern.pdf` e tire as suas conclusões!

4. O Polinómio é nulo?
Numa determinada aplicação é dado um polinómio de coeficientes inteiros, que não está necessariamente na forma normalizada (soma de monómios), por exemplo

$$((4x^3 - 8x^2 - 18)^3 - (-22x^4 + 29x^3 + 2)^3)(x - 12x^2)^2 + 125(x^8 - 3x^6 + 6)$$

Pretende-se saber se o polinómio é ou não identicamente nulo. Um método de o saber é trivial: efectuar os produtos e calcular as potências por forma a que o polinómio fique na forma normal. Se todos os coeficientes de todos os monómios forem nulos a resposta é SIM , caso contrário é NÃO.

Este método leva (no pior caso) um tempo exponencial, uma vez que o tamanho da forma expandida (normal) do polinómio pode ser exponencial no tamanho do polinómio original.

Desenvolva um algoritmo (determinístico) eficiente (não exponencial) para resolver este problema. Para isso, calcule o valor do polinómio para determinados valores inteiros de x e use o chamado "Teorema Fundamental da Aritmética": um polinómio de coeficientes reais de grau n tem no máximo n raízes reais. Suponha que o grau do polinómio é polinomial relativamente ao comprimento do polinómio.

Pressupõe-se que as seguintes operações são "eficientes":

– Determinação do grau do polinómio dado.
– Cálculo do polinómio (na forma dada) para um valor de x.