Fondamenti di Informatica, A.A. 2013-2014

01/09/2014

Esercizio 1

È dato il frammento di codice Matlab

```
n = 4;
v = [1  3  2  4];
m = 1;
for  i=1:n
   v(i) = v(i)^3
   [m  j] = min(v)
   v(i) = m+v(i)
end
disp(v);
```

Soluzione

Al primo passo il primo elemento del vettore prende il valore 2; rimane poi sempre il più piccolo, e quindi il risultato è dato da 2 aggiunto alla somma dei cubi degli altri elementi, ossia

```
octave:5> disp(v);
     2    29    10    66
```

Esercizio 2

Descrivere la aritmetica *floating-point* usata in Matlab

Soluzione

Su calcolatore si utilizzano i numeri floating-point: sono un sottoinsieme dei numeri reali (razionali) della forma (s, e, f) dove s è il segno, e è l'esponente e f è la parte frazionaria, o mantissa:

$$(s, e, f) = s \times f \times \beta^{e-t} = \pm\beta^e \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots \frac{d_t}{\beta^t} \right)$$

ed f è rappresentato su t cifre

$$.d_1 d_2 \dots d_t$$

L'idea sottostante ai numeri floating-point è mutuata dalla notazione scientifica, ad esempio $x = 2300 = 2.3 \times 10^3 = 0.23 \times 10^4$. Si assume di solito che in numeri siano normalizzati: $m \geq \beta^{t-1}$, ovvero $d_1 \neq 0$; all'insieme dei numeri normalizzati si aggiunge esplicitamente lo 0.

Un numero reale a non è in generale rappresentabile esattamente. Se $\mathrm{fl}(a)$ è il numero in virgola mobile che meglio approssima il numero reale a, si definisce la precisione di macchina tramite la espressione

$$\mathrm{fl}(a) = a(1 + \epsilon), \qquad |\epsilon| \leq u,$$

dove la quantità

$$u = \frac{1}{2}\beta^{1-t}$$

è detta arrotondamento unitario. In Octave/Matlab si usa l'aritmetica IEEE 754:

Singola precisione $\beta = 2$, $t = 23 + 1$ (bit "fantasma" se normalizzato 1, se no 0), $e_{min} = -127$, $e_{max} = 127$, $u = 6 \times 10^{-8}$, range $10^{\pm 38}$

Doppia precisione $\beta = 2$, $t = 52 + 1$ (bit "fantasma" se normalizzato 1, se no 0), $e_{min} = -1023$, $e_{max} = 1023$, $u = 1 \times 10^{-16}$, range $10^{\pm 308}$

ed in particolare, Matlab/Octave usano esclusivamnte la doppia precisione.

Le operazioni IEEE verificano

$$u \oplus v = \text{round}(u + v)$$

ossia il risultato della singola operazione in virgola mobile è eguale all'arrotondamento del risultato esatto. Per le operazioni in virgola mobile, a causa dell'arrotondamento, cadono le proprietà di associatività e distributività della aritmetica sui numeri reali.

Esercizio 3

Il seguente codice viene proposto per effettuare la ricerca lineare di un elemento in un vettore. Il codice contiene un errore ed una inefficienza; si identifichino e si proponga una soluzione.

```
function pos=lseach(key,v)
for k=1:length(v)
    if (key==v(k))
        pos=k;
    end
end
```

Soluzione

L'errore consiste nella mancata assegnazione del valore `pos` nel caso in cui la chiave `key` non sia presente nel vettore. L'inefficienza consiste nel fatto che anche quando la chiave venga trovata, si prosegue comunque nell'esame del vettore fino al termine; è da notare che qualora il vettore contenga più elementi eguali alla chiave, questa versione del codice restituisce la posizione dell'*ultimo*.

Una possibile modifica del codice usa il valore 0 per segnalare chiave non presente:

```
function pos=lseach(key,v)
pos = 0;
for k=1:length(v)
    if (key==v(k))
        pos=k;
        break;
    end
end
```

Esercizio 4

È data la seguente funzione Matlab

```
function [z]=rconjg(a)
  [n,nc]=size(a);
  if (n ~= nc)
    error('Sizes must be equal');
  end
  z=eye(n);
  p=zeros(n,1);
  for i=1:n
    for j=i+1:n
      p(j) = a(i,:)*z(:,j);
      z(:,j) = z(:,j) - (p(j)/p(i))*z(:,i);
    end
  end
```

Si stimi il numero di operazioni aritmetiche eseguite.

Soluzione

Nel ciclo più interno abbiamo un prodotto scalare tra due vettori di dimensione n per un costo $2n$, seguito da una somma di un vettore con un multiplo di un secondo vettore, per un costo di $2n+1$, avendo considerato anche il costo della divisione tra i due scalari `p(j)/p(i)`. In definitiva abbiamo

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=i+1}^{n}(2n+2n+1) &= \sum_{i=1}^{n}\left((4n+1)\cdot\sum_{j=i+1}^{n}1\right)=\sum_{i=1}^{n}(4n+1)(n-(i+1)+1)= \\ &= (4n+1)\sum_{i=1}^{n}(n-i)=(4n+1)\sum_{j=0}^{(n-1)}j= \\ &= \frac{(4n+1)(n-1)n}{2}\approx 2n^3\end{aligned}$$

avendo usato la formula $\sum_{k=0}^{m}=m(m+1)/2$ ed osservando che quando i varia da 1 a n, allora $n-i$ varia da 0 a $n-1$.

Esercizio 5

È dato il frammento di codice Matlab

```
i=1;
s1=0;
s2=0;
while (i<=length(v))
    if (mod(i,2)==1)
        s1 = s1 + sqrt(v(i));
    else
```

```
        s2 = s2 - sqrt(v(i));
    end
    i = i + 1;
end
```

Riscrivere il codice usando uno o più cicli `for` e senza usare la istruzione `if`. Bonus: riscrivere usando solo operazioni su array.

Soluzione

Soluzione con cicli `for`

```
s1=0;
s2=0;
for i=1:2:length(v)
  s1 = s1 + sqrt(v(i));
end
for i=2:2:length(v)
  s2 = s2 - sqrt(v(i));
end
```

Soluzione con operatori su array:

```
s1 =  sum(sqrt(v(1:2:end)));
s2 = -sum(sqrt(v(2:2:end)));
```