Un algoritmo di arrotondamento...

Tratteremo un algoritmo di arrotondamento poco usuale ma molto utile evidenziando, più che l'algoritmo stesso, il percorso che ha portato a questa soluzione. Si cercherà di dimostrare come a volte, durante lo sviluppo di un programma, può essere opportuno fermarsi a riflettere per scegliere la strada migliore anzichè iniziare subito a scrivere del codice,.

Una delle operazioni che sicuramente ogni programmatore ha dovuto, prima o poi, affrontare durante la stesura di un programma è certamente la necessità di arrotondare dei numeri; tanto che tutti i linguaggi di programmazione prevedono una funzione di arrotondamento del tipo:

```
x=int(3.2)
```

dove il numero 3,2 (scelto come esempio assolutamente a caso) viene normalmente arrotondato per difetto assegnando alla variabile x il valore 3.
Un effetto similare si può ottenere, per i linguaggi tipizzati, eseguendo il cast da un numero reale (ad es. un double) ad un intero:

```
x=(int) 3.2
```

Alcuni linguaggi più evoluti prevedono forme distinte per l'arrotondamento verso il basso o l'alto; per esempio in java:

```
x=floor(3.2) assegna a x il valore 3
x=ceil(3.2) assegna a x il valore 4
```

Ove non esplicitamente indicato tramite funzioni specifiche l'arrotondamento viene usualmente eseguito verso il basso con i decimali che terminano con il 5 (incluso) e verso l'alto per decimali he teminano con il 6. Quindi ad esempio:

```
x=(int) 3.3 => x=3
x=(int) 3.5 => x=3
x=(int) 3.7 => x=4
```

Le stesse regole si hanno per funzioni ove viene richiesto l'arrotondamento al decimale ennesimo ovvero con numeri che vengono arrotondati a *n* decimali dopo la virgola.

Ogni programmatore sa, inoltre, sfruttare queste funzioni per ottenere arrotondamenti a multipli di 10. Per esempio dato un numero

```
x=321
```

per arrotondare a 330 sono sufficienti le seguenti operazioni (dove il valore di x dopo l'assegnazione è mostrato fra parentesi):

```
x=x/10          (32.1)
x=x+1           (33.1)
x=int(x)        (33)
x=x*10          (330)
```

o più semplicemente:

```
x=int((x/10)+1)*10
```

Il gioco si può ripetere sostituendo al 10, il 100, il 1000 etc a seconda dell' arrotondamento desiderato.
Si noti che gli esempi sopra riportati sono espressi il più possibile con formule matematiche o metalinguaggio non volendo in portare in questo contesto esempi specifici di alcun linguaggio di programmazione.

La cosa si complica notevolmente se l'arrotondamento non può essere predeterminato al momento della scrittura del programma e deve essere determinato in risposta al valore immesso da un utente o calcolato su dati variabili.
L'algoritmo che viene presentato nasce infatti dalla necessità, richiesta in un programma che la scala di un grafico fosse arrotondata alla decina, centinaio, migliaio (etc.) successivo in funzione dei dati da rappresentare.
Per chiarire per valori da 1 a 10 la scala doveva essere arrotondata a all'unità successiva al valore massimo da rappresentare nel grafico; per valori da 10 a 100 la scala doveva essere arrotondata alla decina successiva al valore massimo, e così via.

In prima analisi la soluzione più istintiva prevedeva l'uso di una serie di if per determinare in quale range si trovava il valore inserito

```
es:   if (vmax > 10 AND vmax <= 100)
```

e per ciascun range il relativo arrotondamento.

Questa soluzione, scartata immediatamente, presenta l'inconveniente di rendere il codice molto prolisso e non risolve completamente il problema in quanto non essendo il valore da rappresentare predeterminato dovremmo scrivere una serie paticamente infinita di condizioni IF (almeno fino al limite fornito dal linguaggio di programmazione per il tipo di numero rappresentato).

Una soluzione più evoluta poteva consistere nella conversione del numero in striga e nel suo trattamento secondo convenienza come si cerca di esprimere nei seguenti passi descritti senza usare un linguaggio specifico:

1. trasformazione del numero in stringa es.321 -> "321"
2. estrazione della cifra più significativa (a sx) "3"
3. memorizzazione del numero di cifre rimanenti "21" => 2 cifre
4. riconversione della nuova stringa in numero "3" -> 3
5. somma di una unità 3+1=4
6. riconversione in stringa 4 -> "4"
7. aggiunta di un numero di zeri pari alle cifre rimanenti "400"
8. ritrasformazione in numero

La soluzione ha un numero di passi finiti ma non è sicuramente molto elegante; inoltre molti linguaggi non facilitano il trattamento di stringhe per cui le varie conversioni sopra riportate possono diventare a dir poco difficoltose a seconda del linguaggio utilizzato. Osservando la soluzione ipotizzata, per quanto non ideale essa ha certamente il valore di evidenziare che la soluzione consiste nel individuare la cifra più significativa e il numero di cifre successive ovvero la potenza di 10 che rappresenta il corretto range di valori.

In tempi andati i programmatori cercavano spremere ogni singolo ciclo macchina dai processori ed i bytes erano preziosi per cui in casi di questo tipo si aveva l'abitudine di verificare soluzioni alternative di tipo 'matematico' che potessero aumentare la velocità del programma o perlomeno riducessero lo spazio occupato dal codice. Per cui, rispolverando vecchie nozioni di matematica, magari con l'aiuto di un foglio di calcolo per le verifiche, ci si può rendere conto che il Logaritmo in base 10 (Log10) di un numero contiene nella sua parte intera un numero che rappresenta il multiplo di 10 nel cui range è contenuto il valore in esame. Visto che un esempio vale più di mille parole ecco il Log10 di alcuni valori:

Val	Log10(Val)	Int(Log10(Val))
11	1,0413927	1
156	2,	2
289	2,4608978	2
1233	3,0909631	3
56890	4,7550359	4

Da qui il passo è relativamente facile:

Se si eleva 10 per il numero ricavato si ottiene, all'inverso la potenza di 10, arrotondata per difetto, più vicina al numero iniziale. Tornando agli esempi:

x	Log10(x)	Int(Log10(x))=y	10Exp y=z
11	1,0413927	1	10
20	1,30103	1	10
156	2,1931246	2	100
289	2,4608978	2	100
1233	3,0909631	3	1000
56890	4,7550359	4	10000

Per brevità èstato denominato **y** il numero ricavato da Int(Log10(x)) e d'ora in avanti **z** sara 10 Exp y (ovvero 10 elevato alla y).

Da qui se dividendo il numero iniziale per z e sommando 1 si otterrà per ecesso la base del numero cercato; es:

```
289/100=2.89 + 1 =3.89
```

Da cui arrotondando e rimoltiplicando per z:

```
Int(3.89)=3*100=300
```

ecco l'arrotondamento cercato.

Per verifica ecco gli arrotondamenti di tutti i numeri usati come esempi:

x	Log10(x)	y	z	arrot.
11	1,0413927	1	10	20
20	1,30103	1	10	30
156	2,1931246	2	100	200
289	2,4608978	2	100	300
1233	3,0909631	3	1000	2000
56890	4,7550359	4	10000	60000

La formula matematica completa che esprime il tutto è quindi:

```
x=int((Val/ 10Exp( Int(Log10(Val)) ) )+1) * 10Exp(Int(Log10(Val)))
```

Come si vede una sola formula matematica può sostituire decine di righe di condizioni if-else e relativi calcoli o un certo numero di conversioni e riconversioni. Semplice, elegante e utile...

Da notare che non tutti i linguaggi permettono questo calcolo in una sola riga. Nel caso del programma da cui era nata la necessità, essendo sviluppando in Java ci si scontrava con il problema che tale linguaggio non possiede la funzione di elevazione a potenza che permettesse il calcolo 10Exp n quindi prima è stato necessario costruire una funzione che permettesse questo calcolo e solo in seguito è stato possibile eseguire la formula per calcolare l'arrotondamento desiderato...

Nota finale: l'algoritmo di cui sopra è descritto in maniera scientifica, nella sua forma più generale in wikipedia ove non risulta ne facile da trovare ne molto intuitivo essendo descritto in puri termini matematici. Spero quindi che la spiegazione pratica quì presentata trovi il dovuto apprezzamento.

di Rudi Giacomini Pilon