a cura di Francesco Romani

Calcolo enigmatico con *Mathematica*

Tanto per alleggerire l'atmosfera dopo gli articoli di Fisica, Matematica ed Informatica Teorica (e ne arriveranno altri) mi sono trovato un collaboratore esperto di enigmistica con cui presenteremo una serie di articoli sulla soluzione con Mathematica dei più noti giochi enigmistici. Questo mese tocca al "Calcolo Enigmatico".

di Federico Curcio e Francesco Romani

1. Il Problema

Affrontiamo un tipico gioco presente in ogni numero de *La Settimana Enigmistica*. Dato un insieme di equazioni aritmetiche ove le cifre sono state sostituite con altrettanti simboli, il gioco consiste nel trovare l'unica corretta associazione simboli-cifre che soddisfi tutte le equazioni, deducendola da esse.

Alcune cifre sono talvolta facili da associare al relativo simbolo. Ad esempio, ponendo che * significhi "qualsiasi simbolo (anche assente)", da una equazione del tipo:

$$**\clubsuit - **\clubsuit = **\nabla$$

se ne deduce che ∇=0; infatti la cifra meno significativa del risultato di una sottrazione fra due numeri aventi uguali le cifre meno significative non può che essere lo zero. Ancora, da:

$$\spadesuit + \blacklozenge = \otimes *$$

se ne deduce che $\otimes$=1; infatti la somma di due singole cifre non può mai essere maggiore di 18 e quindi, se il risultato prevede due cifre, la più significativa non può che essere l'uno.

Più spesso è solo possibile escludere alcuni valori dei simboli, come ad esempio in:

$$**\Diamond \times **\Diamond = **\infty$$

che ci fa eliminare le cifre 0, 1, 5 e 6 per $\Diamond$: infatti il loro prodotto darebbe come cifra meno significativa la stessa cifra di partenza, il che, nel caso dell'esempio, non accade. Per quanto riguarda ∞, esso può valere solo 1, 4, 6 oppure 9, che sono le cifre meno significative che si ottengono moltiplicando per se stesse le cifre non escluse per $\Diamond$, cioè 2, 3, 4, 7, 8 e 9.

Capita anche di imbattersi in equazioni che permettono di stabilire quale, fra due simboli, abbia il valore associato maggiore. Da:

$$\nabla\spadesuit\clubsuit - \blacklozenge = \nabla\heartsuit *$$

si deduce che, essendo cambiato il simbolo associato alla cifra delle decine in conseguenza di una sottrazione di un valore minore di 10, sussiste la relazione $\spadesuit > \heartsuit$; in questo caso possiamo aggiungere che: $\spadesuit = \heartsuit + 1$.

Con ragionamenti analoghi applicati a tutte le equazioni dello schema di gioco, si individuano via via le varie cifre fino a trovare la soluzione.

Forse ad alcuni lettori già appare evidente che la risoluzione di questo gioco ben si presta ad essere automatizzata.

L'approccio che utilizziamo in questo articolo è quello brutale. Esaminiamo tutte le possibili associazioni simbolo-cifra, considerando che "a simbolo uguale corrisponde cifra uguale" e che si escludono associazioni multiple. Poiché ad un simbolo corrisponde una e una sola cifra, i tentativi da effettuare sono pari alle permutazioni di 10 elementi cioè 10! = 3.628.800. Ciascun tentativo prevede l'analisi delle singole equazioni e la prosecuzione col successivo nel caso anche una sola non fosse soddisfatta. L'implementazione di questo tipo di soluzione del problema (anche se meno raffinata di altre) permette di sperimentare un po' di interessante programmazione combinatoria.

Ricordiamo infine l'approccio meno faticoso: attendere l'uscita del successivo numero del periodico enigmistico e sbirciare la soluzione in quarta di copertina. Sempre che si abbia pazienza.

Un problema molto più complesso, che non trattiamo qui, è

quello di trovare un problema che abbia esattamente una sola soluzione. In questo caso l'approccio brutale porta poco lontano ed è preferibile un approccio strutturato... lo lasciamo come esercizio ai lettori.

2. Rappresentazione del problema

Vediamo innanzitutto come rappresentare e trattare il problema. Un esempio tipico consiste in una matrice 5X5 composta di gruppi di simboli e operazioni aritmetiche e segni di uguaglianza. Trascriviamone una, scrivendo lettere dell'alfabeto al posto dei simboli. Con *Mathematica* 3.0 la matrice di ingresso può essere editata direttamente.

In[1]:=

$$\text{Mat1} = \begin{pmatrix} abc & "+" & def & "==" & bfe \\ "-" & "" & "/" & "" & "-" \\ gh & "*" & b & "==" & eih \\ "==" & "" & "==" & "" & "==" \\ acc & "-" & fl & "==" & ile \end{pmatrix}$$

Out[1]=

$$\begin{pmatrix} abc + def == bfe \\ - \quad / \quad - \\ gh * b == eih \\ == \quad == \quad == \\ acc - fl == ile \end{pmatrix}$$

La funzione **explode** trasforma un simbolo di più caratteri nel numero in base 10 che ha per cifre le lettere del simbolo, si ignora tutto ciò che non è simbolo in modo da poter applicare **explode** a strutture complesse con la funzione **MapAll** (abbreviata **//@**) che applica una funzione ad una lista e a tutte le sottoliste.

In[2]:=

```
explode[s_Symbol]:=
 Module[{l},
  l=ToExpression[
    Characters[ToString[s]]];
  l.Reverse[
   10^(Range[Length[l]]-1)]];
explode[x_]:=x;
```

In[3]:=

```
explode[abc]
```

Out[3]=

```
100 a+10 b+c
```

In[4]:=

```
explode//@Mat1
```

Out[4]=

$$\begin{pmatrix} 100a+10b+c + 100d+10e+f == 100b+e+10f \\ - \quad / \quad - \\ 10g+h * b == 100e+h+10i \\ == \quad == \quad == \\ 100a+11c - 10f+1 == e+100i+10l \end{pmatrix}$$

La funzione **ts** trasforma una lista in una stringa. Applicandola alle righe 1, 3, 5 e alle colonne 1, 3, 5 della matrice e tornando alle espressioni con **ToExpression** si scrivono le equazioni del problema.

In[5]:=

```
ts[x_]:=StringJoin@@ToString/@x
toEq[M_]:=ToExpression[Join[
  ts/@M[[{1,3,5}]],
  ts/@(Transpose[M][[{1,3,5}]])]];
```

In[6]:=

```
toEq[Mat1]//ColumnForm
```

Out[6]=

```
abc + def == bfe
b gh == eih
acc - fl == ile
abc - gh == acc
def/b == fl
bfe - eih == ile
```

Esplodendole si ottengono le equazioni in termini dei singoli simboli.

In[7]:=

```
eq10=explode//@toEq[Mat1];
eq10//ColumnForm
```

Out[7]=

```
100a+10b+c+100d+10e+f == 100b+e+10f
b(10g+h) == 100e+h+10i
100a+11c-10f-1 == e+100i+10l
100a+10b+c-10g-h == 100a+11c
(100d+10e+f)/b == 10f+1
100b-99e+10f-h-10i == e+100i+10l
```

La congiunzione di tutte le equazioni è la condizione logica che deve essere soddisfatta.

In[8]:=

```
eqtest=And@@eq10
```

Out[8]=

```
100a+10b+c+100d+10e+f  ==  100b+e+10f  &&
b(10g+h) == 100e+h+10i && 100a+11c-10f-1 ==
e+100i+101 && 100a+10b+c-10g-h == 100a+11c &&
(100d+10e+f)/b == 10f+1 && 100b-99e+10f-h-10i
== e+100i+101
```

3. Calcolo delle permutazioni

Il nostro problema è adesso applicare questo test a tutte le permutazioni delle cifre da 0 a 9. Il calcolo di tutte le permutazioni di n oggetti è implementato dalla funzione **Permutations** ma 10! vettori di 10 elementi fanno almeno 36 Megabytes di memoria, troppi anche per una macchina robusta.

Vediamo un programma ricorsivo che calcola le permutazioni.

In[1]:=

```
perm[a_]:=perm[{},a];
perm[a_,{}]:=AppendTo[LL,a];
perm[a_,b_]:=
   Scan[perm[Append[a,#],
        Complement[b,{#}]]&,b]
```

In[2]:=

```
LL={};
perm[{1,2,3}];
LL
```

Out[2]=

```
{{1,2,3},{1,3,2},{2,1,3},{2,3,1},{3,1,2},{3,2
,1}}
```

Per calcolare le permutazioni di un vettore di 7 elementi bastano 0.216 secondi alla funzione *builtin* e ne servono 110 a quella fatta in casa.

In[3]:=

```
Timing[Permutations[Range[7]];]
```

Out[3]=

```
{0.216667 Second,Null}
```

In[4]:=

```
Timing[LL={};perm[Range[7]];]
```

Out[4]=

```
{110.717 Second,Null}
```

Il vantaggio della funzione fatta in casa è la facile modificabilità: invece di appendere le permutazioni alla lista **LL** ci applichiamo la funzione **t**.

In[5]:=

```
perm[a_,{}]:=t[a];
```

```
Timing[perm[Range[7]];]
```

Out[6]=

```
{10.2833 Second,Null}
```

In[7]:=

```
Timing[perm[Range[8]];]
```

Out[7]=

```
{82.7833 Second,Null}
```

Andando avanti così ci aspettiamo di impiegare 80×9×10 secondi per applicare **t** alle 10! permutazioni che ci interessano.

Implementiamo **t[v]** in modo che fermi la ricorsione e "lanci" fuori il vettore giusto quando lo trova. Per ora ci limitiamo a cercare un vettore dato. Si noti l'uso di **Throw** per lanciare il risultato e di **Catch** per prenderlo al volo. Si noti anche l'uso del predicato **SameQ** (abbreviato con **===**) che forza il test a **True** o **False**.

In[8]:=

```
t[v_]:=If[a==={1,3,2,5,4,6,7,8},Throw[a]]
Timing[Catch[perm[Range[8]]]]
```

Out[8]=

```
{1.66667 Second,{1,3,2,5,4,6,7,8}}
```

Se cerchiamo nel pagliaio sbagliato il vettore non viene trovato mai e si misura il tempo di tutte gli 8! tentativi.

In[9]:=

```
Timing[Catch[perm[Range[8]-1]]]
```

Out[9]=

```
{89.2667 Second,Null}
```

Per rendere il calcolo più efficiente possiamo memorizzare le permutazioni di 7 elementi e fermare molto prima il processo di ricorsione.

In[10]:=

```
per[b_]:=per[{},b];
per[a_,b_]:=
  Scan[test[Join[a,b[[#]]]]&,p7]/;
    Length[b]==7
per[a_,b_]:=
   Scan[per[Append[a,#],
        Complement[b,{#}]]&,b]
```

Finché **test** rimane indefinito si misura solo il tempo del calcolo delle permutazioni.

In[11]:=

```
Timing[Catch[per[Range[8]]]]
```

Out[11]=

```
{11.4333 Second,Null}
```

In[12]:=

```
Timing[Catch[per[Range[9]]]]
```

Out[12]=

```
{105.2 Second,Null}
```

Ora ci aspettiamo di impiegare 105×10 secondi per applicare **test** alle 10! permutazioni che ci interessano: abbiamo guadagnato un fattore 10.

4. Ricerca della soluzione

Ora non resta che implementare **test**...

In[1]:=

```
test[list_]:=(
  {a,b,c,d,e,f,g,h,i,l}=list;
  If[eqtest,Throw[list]]);
```

... e lanciare il missile.

In[2]:=

```
Clear[a,b,c,d,e,f,g,h,i,l]
Timing[Catch[per[Range[10]-1]]]
{1973.08 Second,{3,9,1,5,7,6,8,0,2,4}}
```

Scriviamo la soluzione (tenendo conto che al momento del **Throw** i valori di **a,b,...,l** sono quelli giusti) ...

In[3]:=

```
explode//@Mat1
```

Out[3]=

$$\begin{pmatrix} 391 & + & 576 & == & 967 \\ - & & / & & - \\ 80 & * & 9 & == & 720 \\ == & & == & & == \\ 311 & - & 64 & == & 247 \end{pmatrix}$$

... e verifichiamola.

In[4]:=

```
toEq[explode//@Mat1]
```

Out[4]=

```
{True,True,True,True,True,True}
```

Tanto per divertirci ancora proviamo con il problema del numero successivo del settimanale:

In[5]:=

```
Clear[a,b,c,d,e,f,g,h,i,l]
```

In[6]=

$$\mathbf{Mat} = \begin{pmatrix} \mathbf{ab} & "*" & \mathbf{b} & "==" & \mathbf{cde} \\ "+" & "" & "-" & "" & "-" \\ \mathbf{ddf} & "/" & \mathbf{a} & "==" & \mathbf{gb} \\ "==" & "" & "==" & "" & "==" \\ \mathbf{dbg} & "-" & \mathbf{c} & "==" & \mathbf{dbd} \end{pmatrix}$$

In[7]:=

```
eq10=explode//@toEq[Mat];
eqtest=And@@eq10;
Clear[a,b,c,d,e,f,g,h,i,l]
Timing[Catch[per[Range[10]-1]]]
```

Out[7]=

```
{2179.83 Second,{4,7,3,2,9,8,5,0,1,6}}
```

In[7]:=

```
explode//@Mat2
```

Out[8]

$$\begin{pmatrix} 47 & * & 7 & == & 329 \\ + & & - & & - \\ 228 & / & 4 & == & 57 \\ == & & == & & == \\ 275 & - & 3 & == & 272 \end{pmatrix}$$

In[9]:=

```
toEq[explode//@Mat2]
```

Out[9]

```
{True,True,True,True,True,True}
```

5. Rappresentazione del problema con i simboli

Dimenticavamo... qualcuno potrebbe voler rappresentare il problema usando simboli strani invece delle più prosaiche lettere.

Con la versione 3.0 di *Mathematica* si possono usare le *Palette* di input.

In **Figura 1** vediamo una porzione di quella che contiene tutti i caratteri (**Complete Characters** nel sottomenu **Palettes** del menu **File**).

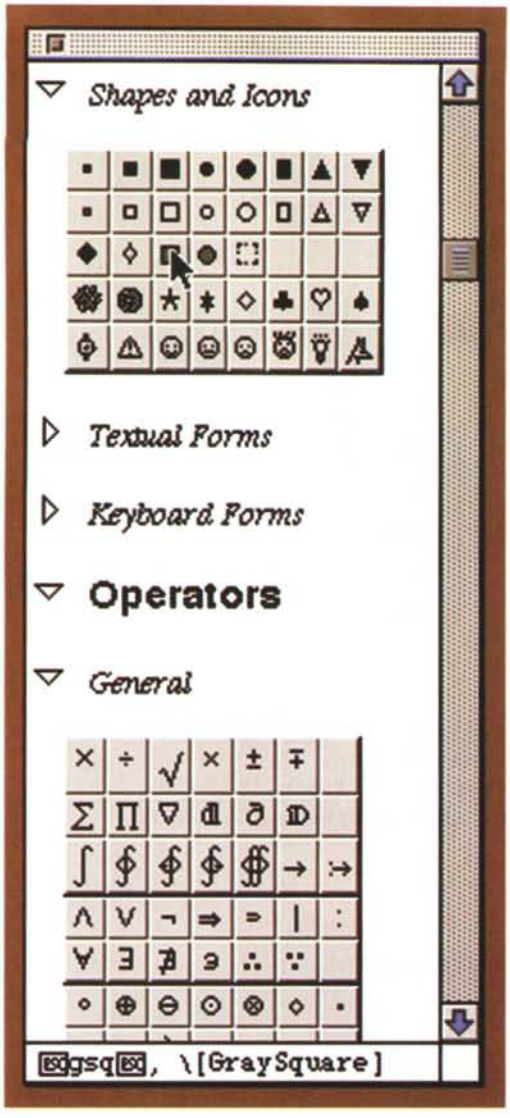


Figura 1

Cliccando un simbolo, questo viene scritto nel Notebook nel punto dove è la selezione. Passando sopra i simboli si vedono in basso le sequenze di caratteri per un input manuale. Per esempio il quadratino grigio può essere scritto con **\[GraySquare]** oppure con **gsq** preceduto e seguito dal tasto **ESCAPE**. Usando la palette si può scrivere una serie di regole di conversione:

In[1]:=

```
conv = {
   "==" -> "=", "/" -> ":",
   "*" -> "×", "a" -> "■",
   "b" -> "●", "c" -> "□",
   "d" -> "❀", "e" -> "◊",
   "f" -> "♣", "g" -> "♡",
   "h" -> "♠", "i" -> "☺",
   "l" -> "■"};
```

Definiamo una funzione che applica le regole di conversione alla matrice di input:

In[2]:=

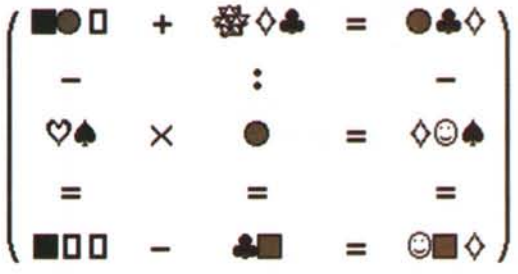
```
simboli[x_Symbol]:=
 StringReplace[ToString[x],conv];
simboli[x_]:=x;
```

Vediamo ora il primo problema ...:

In[3]:=

```
simboli//@Mat1
```

Out[3]=

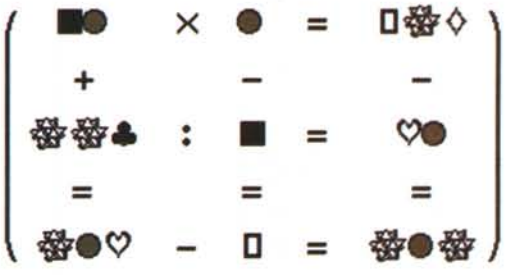


... e il secondo:

In[4]:=

```
simboli//@Mat2
```

Out[4]=

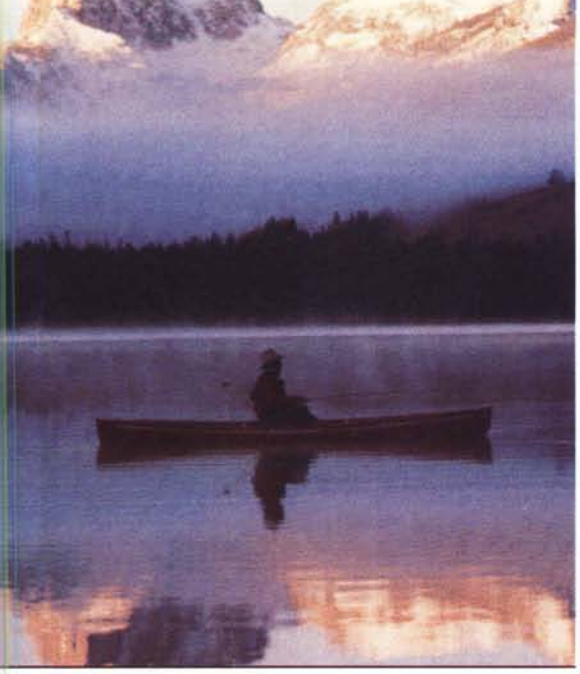


MC

Bibliografia

La Settimana Enigmistica, n. 3414, 30 Agosto 1997, problema n. 1452, pag. 12.

La Settimana Enigmistica, n. 3415, 6 Settembre 1997, problema n. 1555, pag. 12.



C'e' chi ha deciso di isolarsi ma .....
..... ma per te che hai fame di veloci connessioni c'e'

CoFax
TELEMATICA

http://www.cofax.it
Roma: 06/58201362
Milano: 02/29526100

ISDN point

AVM

prodotti leader del mercato ISDN in Germania.

FRITZ! CARD

ZyXEL

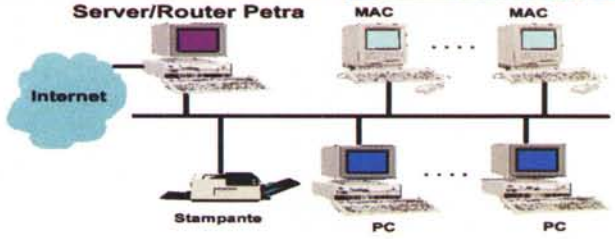
Router, modem, TA: tutti i prodotti ISDN per i vostri PC e la vostra rete.

SEDLBAUER AG

Schede ISDN per tutti i gusti !

Petra
Internet Gateway

Gateway, router, fax: incredibile rapporto prezzo/prestazioni !



Ed inoltre ...

Caldera OpenLinux

La piu' completa delle distribuzioni Linux (in italiano !).

Max 128

La nuova scheda ISDN per Win 3.x, Win 95 e Win NT 4 !