a cura di Francesco Romani

Giochiamo a Master Mind

Il gioco che trattiamo questo mese è il famoso *Master Mind*. L'implementazione in *Mathematica* della soluzione banale permette di ripassare un po' di semplici tecniche di programmazione. I lettori più raffinati possono cercarsi in letteratura gli algoritmi ottimi e provare per conto loro una implementazione più sofisticata.

di Federico Curcio e Francesco Romani

Il gioco

Il Master Mind è un gioco di induzione che si svolge fra due giocatori: il codificatore ed il decifratore.

Il codificatore sceglie una combinazione di simboli ognuno appartenente al medesimo insieme di simboli, insieme noto anche al decifratore, il quale deve indovinare la combinazione scelta proponendo via via i suoi tentativi, consistenti di sequenze analoghe a quella tenuta segreta, scelta dal codificatore.

Le risposte fornite ai tentativi proposti consistono in una coppia di numeri `{pc,ps}` indicanti i simboli corretti, che sono anche in posizione corretta, e i simboli corretti, che non sono in posizione corretta.

Questo gioco può essere classificato come un problema di ottimizzazione vincolata dinamica, i vincoli essendo le risposte fornite dal codificatore ai tentativi del decifratore.

Ponendo che **s** sia il numero di elementi dell'insieme di simboli, il numero di combinazioni possibili nel caso di combinazioni composte da **c** elementi, con possibilità di ripetizioni di uno o più simboli, è pari a s^c. Lo spazio di ricerca del problema di ottimizzazione è dato dall'insieme delle combinazioni possibili.

La versione commercializzata più diffusa prevede 6 colori come simboli (quindi `s=6`) e combinazioni da 4 simboli (quindi `c=4`); in questo caso le combinazioni, prevedendo ripetizioni, sono $6^4=1296$.

Per individuare la sequenza scelta dal codificatore, il decifratore ha due possibilità: analizzare via via le risposte da un punto di vista logico-matematico (ricorrendo ad un processo induttivo) oppure ricorrere ad un approccio per forza bruta.

L'approccio per forza bruta si basa sull'osservazione che le combinazioni (che chiameremo **candidate**) da provare dopo ogni risposta ad un tentativo sono solo quelle che, confrontate col tentativo stesso, forniscono la medesima risposta: infatti la combinazione da indovinare è certamente compresa fra esse, altrimenti la risposta sarebbe stata differente. Ma fra le potenziali combinazioni candidate, quale proporre come tentativo?

Confrontando una combinazione `t` con tutte le altre si ottengono via via risposte differenti, sotto forma di coppie `{pc,ps}` di numeri che costituiscono la risposta al confronto; ad ogni coppia è possibile associare l'insieme delle combinazioni che l'hanno prodotta nel confronto con la combinazione **t**. Definiamo **risposta peggiore** quella coppia a cui è associato l'insieme più numeroso di combinazioni.

Le possibili risposte ad un tentativo sono 14 in tutto, cioè:

- 4-0: sequenza indovinata
- 3-0: tre simboli al posto giusto, manca il quarto
- 2-0: due simboli al posto giusto, ne mancano due
- 1-0: un solo simbolo al proprio posto
- 0-0: nessun simbolo indovinato
- 2-1: due simboli in posizione giusta e uno in posizione errata
- 2-2: due simboli in posizione giusta e due in posizione errata
- 1-1: un simbolo al posto giusto e uno indovinato
- 1-2: un simbolo al posto giusto e due indovinati
- 1-3: un simbolo al posto giusto e tre indovinati
- 0-1: un solo simbolo indovinato, ma in posizione errata
- 0-2: due simboli indovinati, in posizioni errate
- 0-3: tre simboli indovinati, in posizioni errate
- 0-4: tutti i simboli indovinati, ma in posizioni errate.

Alcune osservazioni:

la risposta 0-0 non è poi così tragica, anzi se il nostro tentativo conteneva più colori diversi fra loro è una risposta molto buona;

la risposta 0-4 è quella solitamente (soprattutto se ottenuta al

primo tentativo) più deleteria dal punto di vista psicologico; infatti il decifratore sa di aver indovinato tutti i colori ma - se inesperto - è facile che si lasci prendere dal panico.

La strategia da adottare è quella di individuare ogni volta - fra le combinazioni candidate - una fra le combinazioni le cui risposte peggiori siano soddisfatte dal minor numero di combinazioni; in questo modo si minimizza ogni volta il numero di combinazioni candidate fra le quali scegliere il successivo tentativo da proporre.

In altre parole, l'algoritmo potrebbe essere formulato nel modo seguente:

- per ogni combinazione candidata, confrontarla con le altre combinazioni candidate, contando le occorrenze di ogni singola risposta possibile;

- alla fine dei confronti, scegliere una delle combinazioni candidate con le migliori risposte peggiori.

Per problemi di dimensioni maggiori un approccio come quello appena mostrato risulta molto pesante dal punto di vista dei tempi di elaborazione, soprattutto per la produzione del primo tentativo.

In tali casi è meglio ricorrere a strategie differenti, come ad esempio gli algoritmi genetici (vedi Bibliografia).

Per il gioco standard (4 gettoni, 6 colori) si dimostra che il numero massimo di tentativi da provare - se viene seguita la strategia ottima - non è superiore a 5 (questo è un buon metodo per valutare qualsiasi algoritmo che si proponga di risolvere il Master Mind).

Rappresentazione del problema

Poniamo di avere 4 gettoni di sei possibili colori. Rappresentiamo un gettone con un carattere da **"A"** a **"F"** e una configurazione come una lista di 4 caratteri.

allc è la lista dei possibili gettoni:

In[1]:=
```
allc={"A","B","C","D","E","F"};
```

ALL la lista di tutte le 1296 configurazioni:

In[2]:=
```
ALL= Flatten[Outer[List,
    allc,allc,allc,allc],3];
{First[ALL],Last[ALL],Length[ALL]}
```

Out[2]=
```
{{A, A, A, A}, {F, F, F, F}, 1296}
```

Definiamo ora le funzioni per calcolare la bontà di una soluzione proposta. **equal** vale **1** se due gettoni sono uguali e **0** altrimenti, **npc** dà il numero degli elementi giusti al posto giusto semplicemente confrontando elemento per elemento:

In[3]:=
```
equal[x_,y_]:=If[x==y,1,0];
npc[a_,b_]:=
  Plus@@MapThread[equal,{a,b}]
```

Più complicato il numero dei colori giusti: **pat** fornisce la lista delle presenze di ogni singolo colore. Si trasforma la lista in un polinomio in 6 variabili, e con un paio di trucchi abbastanza sporchi se ne ricavano i coefficienti:

In[4]:=
```
pat[x_]:=
  First/@List@@Expand[
   Plus@@x+2Plus@@allc]-2
nps[a_,b_]:=
  Plus@@MapThread[
     Min,{pat[a],pat[b]}]
```

la funzione **npp** rende la coppia dei valori **pc** e **ps**:

In[5]:=
```
npp[a_,b_]:={ppc=npc[a,b],nps[a,b]-ppc}
```

Ci farà comodo anche una funzione per estrarre a sorte un elemento da una lista:

In[6]:=
```
sel[x_List]:=x[[Random[Integer,{1,Length[x]}]]];
```

Vediamo un esempio:

In[7]:=
```
sol={"E", "F", "D", "D"}
prova=sel[ALL]
```

Out[7]=
```
{E, F, D, D}
{E, D, E, F}
```

C'è un carattere a posto e due sono i colori indovinati ma fuori posto

In[8]:=
```
npp[sol,prova]
```

Out[8]=
```
{1, 2}
```

Associamo ora ad ogni carattere un colore con i comandi seguenti

In[9]:=
```
col["A"]=Red;
col["B"]=Blue;
col["C"]=Cyan;
col["D"]=Magenta;
col["E"]=Green;
col["F"]=Yellow;
```

Definiamo un rettangolo colorato con un bordo nero e la lettera corrispondente in basso al centro (questo accorgimento è di particolare utilità per i possessori di monitor monocromatici e per i giocatori daltonici):

In[10]:=
```
rect[item_,i_]:=
 {col[item],
  Rectangle[{i, 0}, {i+1, 1}],
  Black,
  Line[{{i, 0}, {i+1, 0},
```

```
        {i+1, 1},{i, 1},{i, 0}}],
  Text[item,{i+0.5,-0.3}]]
```

ShowMM mostra la soluzione proposta restituendo i valori di **pc** e **ps** ottenuti dal confronto con **sol.** I valori sono scritti anche nella **PlotLabel.** Se la soluzione proposta coincide con **sol** viene invece stampato "SOLUZIONE ESATTA".

In[11]:=
```
ShowMM[l_List]:=Module[{c,p,label},
 {pc,ps}=npp[l,sol];
 label=If[pc==4,
   "         SOLUZIONE ESATTA",
   "      pc = "<>ToString[pc]<>
     "    ps = "<>ToString[ps]];
 Show[Graphics[
  Table[rect[l[[i]],i],{i,4}]],
   AspectRatio->1.5/5,
   PlotRange->{{0,5},{-0.5,1}},
   PlotLabel->label];
 {pc,ps}];
```

In[12]:=
```
ShowMM[prova]
```

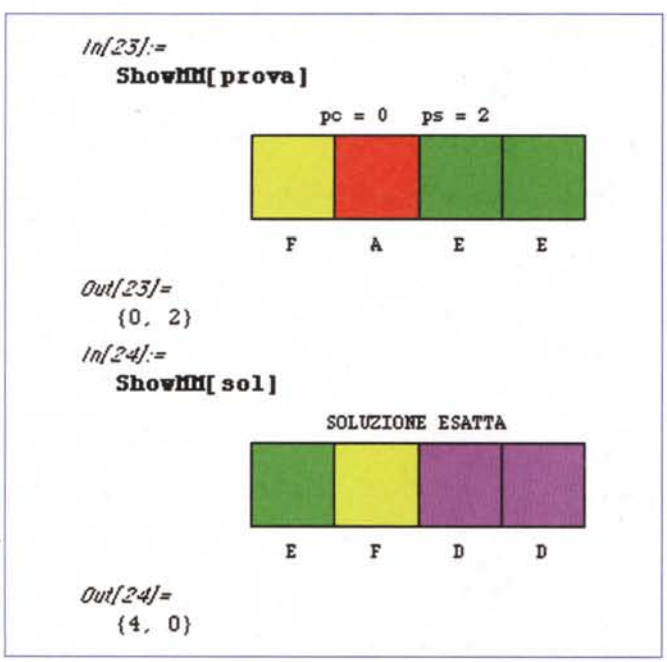

Figura 1

Giocatore umano contro *Mathematica*

Siamo ora pronti a giocare la nostra partita. Il primo caso che trattiamo è quello del giocatore umano contro il *computer*: in questo caso *Mathematica* sostituisce la scatola del gioco e un amico che ci dice quanti gettoni abbiamo azzeccati.

Ci bastano altre due funzioni, una che estragga a caso la soluzione

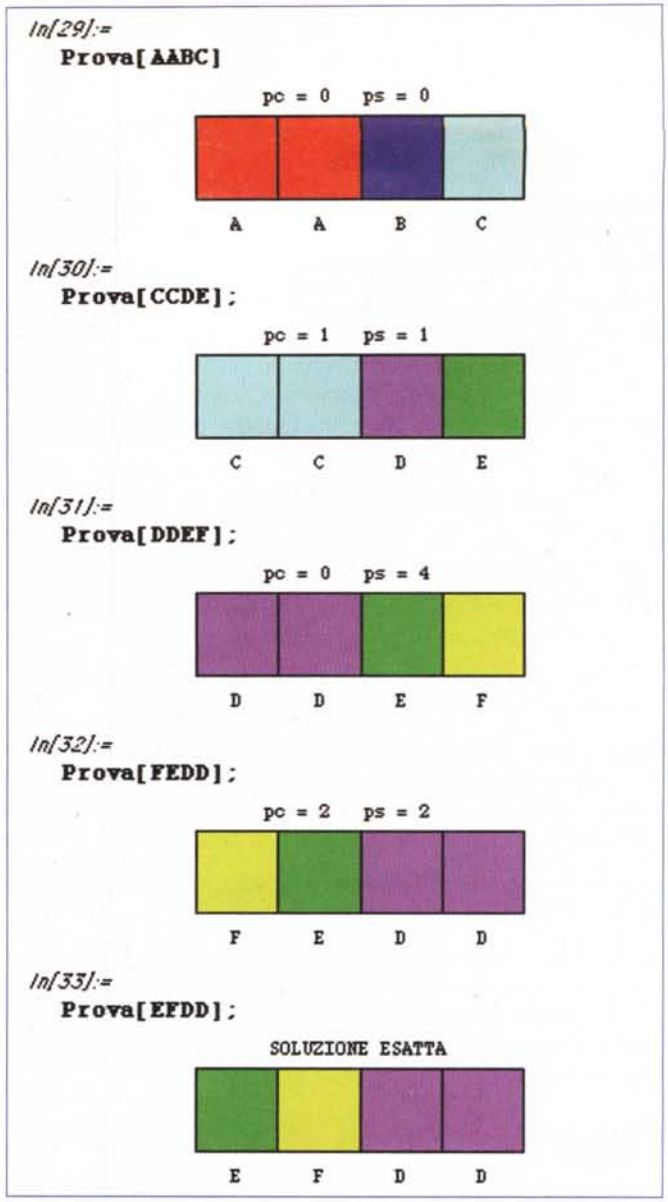

Figura 2

In[1]:=
```
rnd:=Table[sel[allc],{4}]
```

e una per introdurre la nostra proposta

In[2]:=
```
Prova[s_Symbol]:=
 ShowMM[Characters[ToString[s]]]
```

Estraiamo a caso una soluzione e siamo pronti a giocare una partita:

In[3]:=
```
sol=rnd;
```

Vedi Figura 2

Mathematica contro giocatore umano

Più complicato è mettere il *computer* in grado di trovare la soluzione. La strategia che implementiamo è decisamente brutale, ma molto semplice da programmare. Si inizia tenendo nella lista **work** tutte le possibili soluzioni, se ne estrae a sorte una e

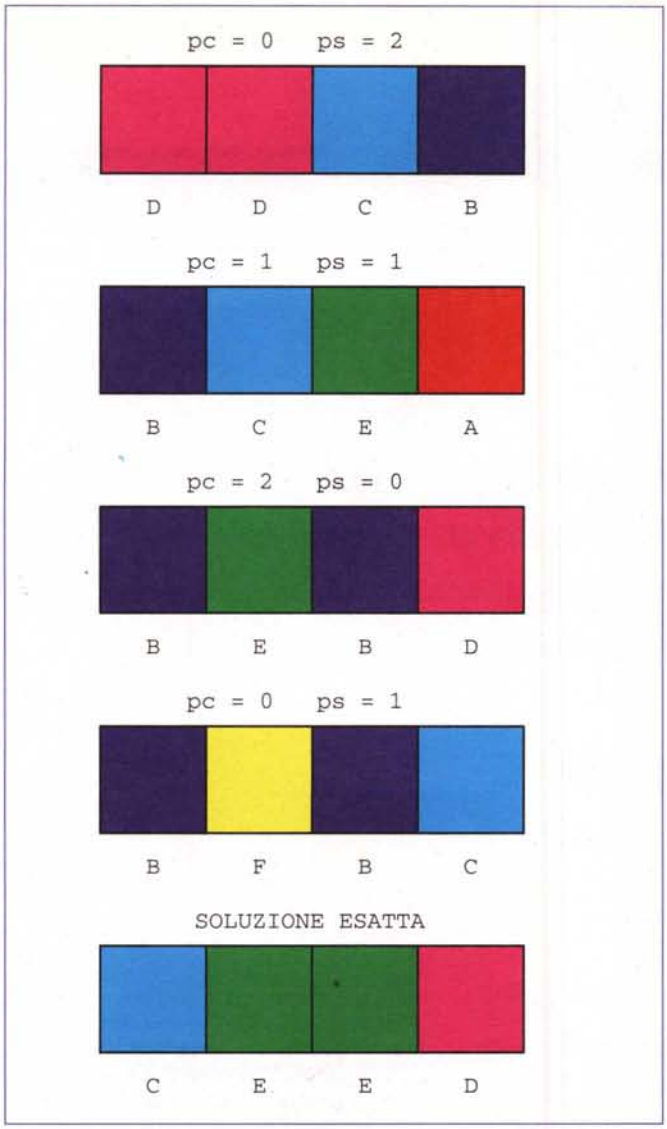

Figura 3

si propone come soluzione. In base alla risposta si selezionano le possibili soluzioni rimaste e si ricomincia.

In[1]:=
```
test[a_,b_]:={npp[a,b],b}
```

In[2]:=
```
riduci:=(
  prova=sel[work];
  vv=ShowMM[prova];
  work=#[[2]]&/@

Select[test[prova,#]&/@work,#[[1]]==vv&];
  l=If[vv=={4,4},0,Length[work]];
  If[l==1,
      ShowMM[First[work]];
      l=0];
  l)
```

In[3]:=
```
gioca:=(
   ngr=1;
   sol=rnd;
   work=ALL;
   While[riduci>0];)
```

In[4]:=
```
gioca
```

Vedi Figura 3

In[5]:=
```
gioca
```

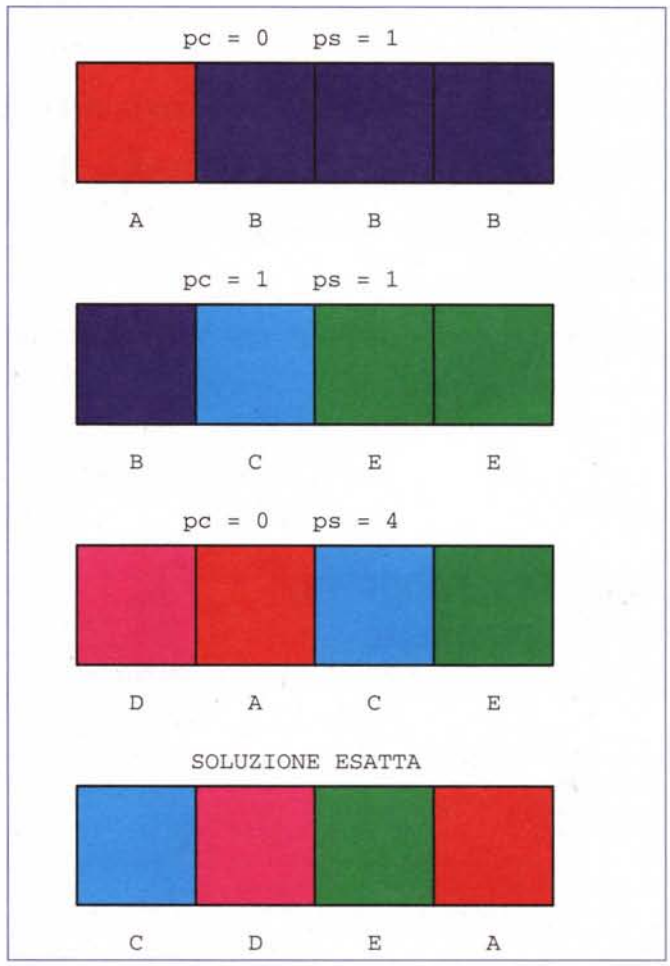

Figura 4

MC

Bibliografia

Corrado Giustozzi **Intelligiochi**, MC nn. 106, 109, 128

D. R. Hope **Master Mind**, in "Personal Software", anno 1, numero 1 (1982)

D. E. Knuth **The computer as Master Mind**, in "Journal of mathematical recreations", n. 9 (1976-77)

K. Koyama e T. W. Lay **An optimal Master Mind strategy** in "Journal of mathematical recreations", n. 25 (1993-94)

P. Zama **Nel mondo del Master Mind**, Sansoni Editore (1984)

J. J. Merelo **Genetic Master Mind: a case of dynamic constraint optimization** (disponibile presso l'autore: jmerelo@kal-el.ugr.es).