a cura di Francesco Romani

LA MACCHINA DI TURING

Tra tutti i vari "modelli" di calcolatore che si sono susseguiti negli anni e tra i vari "linguaggi" di programmazione, la Macchina di Turing ricopre un ruolo a parte. La Macchina di Turing è un "calcolatore" abbastanza potente da poterne simulare qualunque altro e abbastanza semplice da permettere di dimostrare in modo agevole i complessi teoremi della teoria della calcolabilità. D'altra parte programmare una Macchina di Turing può essere considerato un passatempo alla portata anche di persone digiune di informatica

Introduzione

L'idea che sta alla base della Macchina di Turing è quella di una specie di registratore a nastro dotato di un'unità di controllo, una testina e un nastro (di lunghezza illimitata) su cui possono essere scritti simboli di un alfabeto dato ("0", "1", oppure "a", "b", "c" oppure ancora l'alfabeto inglese). All'inizio il nastro contiene i dati di ingresso. L'unità di controllo legge un simbolo, prende una decisione in base ad un insieme finito di regole prefissate, scrive un nuovo simbolo e si sposta di una casella a destra o a sinistra. Sotto particolari condizioni la macchina si ferma e il contenuto del nastro rappresenta il dato in uscita.

Più rigorosamente na Macchina di Turing (MdT nel seguito) è una macchina astratta definita come una quadrupla T = (K,V, A, δ) dove:

K è un insieme finito di stati;

V è l'alfabeto di ingresso che contiene lo speciale simbolo *blank* : "b";

A ∈ K è detto stato iniziale;

δ è una funzione di transizione che associa ad una coppia (p,x) con p∈ K, x∈ V una tripla (q,y,m) con q∈ K, y∈ V, m∈ {destra, sinistra}. Non è richiesto che δ sia definita per tutte le coppie (p,x); comunemente la funzione è data elencando tutte le quintuple (p,x,q,y,m) per cui δ(p,q) è definita.

Si definisce **configurazione** la tripla (q,α,i) dove q è lo stato attuale, α è il contenuto del nastro togliendo i è la posizione del punto di lettura sul nastro (i può assumere qualunque valore intero positivo o negativo).

Si definisce **mossa** il passaggio da una configurazione all'altra, che avviene come segue.

Se lo stato è p, il simbolo in lettura è x e vale δ(p,x)=(q,y,m) allora la nuova configurazione sarà (q,β,j), con β che rappresenta il nastro ottenuto dopo aver sostituito il simbolo in lettura x con

$$j = \begin{cases} i+1 \text{ se } m = \text{destra} \\ i-1 \text{ se } m = \text{sinistra} \end{cases}$$

è la posizione del nuovo punto di lettura.

Se non esiste nessuna quintupla c che inizia per (q, y) l'elaborazione ha termine.

All'inizio si scrive sul nastro una stringa u e si pone i =1 e si entra nello stato A; se al termine la MdT è nella configurazione (q,β,i), il risultato della applicazione della MdT alla stringa u è la stringa v corrispondente alla parte di nastro compresa tra il primo e l'ultimo simbolo non *blank*.

Per definire una MdT basta dare la lista delle quintuple, questo definisce implicitamente l'insieme degli stati e l'alfabeto del nastro (se si trova in lettura un simbolo non presente nelle quintuple la macchina si ferma).

Non è detto che una MdT termini sempre per esempio la macchina definita dalle quintuple:

(A, "0",B,"0",destra);
(B,"0"A,"0",sinistra);

termina senza fare nulla se i primi due simboli della stringa in ingresso sono diversi da "00" e va in ciclo infinito altrimenti.

Quella appena vista è solo una delle tante possibili definizioni del formalismo delle MdT. È però possibile dimostrare che tutte le definizioni sono equivalenti. Cioè, data una coppia A e B di definizioni di formalismo delle MdT, per ogni macchina definita nel modo A è possibile costruire una macchina definita nel modo B che produce lo stesso risultato per ogni stringa di ingresso. Per maggiori dettagli si può consultare l'ottimo libro di Minsky (1967).

Si noti che le MdT, per motivi storici e per l'intrinseca semplicità della loro definizione, rivestono un interesse particolare nel mondo dei formalismi di calcolo per qualche ragione teorica. In questo articolo non siamo interessati alle applicazioni teoriche ma alla simulazione delle MdT con *Mathematica* vediamo quindi subito come scrivere il simulatore. Una trattazione simile (molto più ricca ma meno accessibile ai non esperti) si può trovare nel libro di Maeder (1996).

Il simulatore in *Mathematica*

Vediamo separatamente le varie componenti: il nastro, la funzione di transizione, il simulatore e i programmi di visualizzazione dell'elaborazione.

Nastro

Per simulare un nastro infinito si può usare una funzione che vale "b" su tutti gli interi tranne dove è assegnato esplicitamente un altro valore. **InitTape[s]** piazza sul nastro la stringa di ingresso **s**; **MAXPOS** e **MINPOS** tengono traccia della parte di nastro effettivamente utilizzata.

In[1]:=
```
InitTape[s_String]:=(
    MAXPOS=0;
    Clear[TAPE];
    TAPE[_]:=" ";
    Map[(TAPE[++MAXPOS]=#)&,
          Characters[s]];
    MINPOS=POS=1;)
```

In[2]:=
```
InitTape["1101"]
?TAPE
```

Out[3]=
```
TAPE[1] = "1"
TAPE[2] = "1"
TAPE[3] = "0"
TAPE[4] = "1"
TAPE[_] := " "
```

Quintuple e funzione DELTA

InitDelta[q] definisce la funzione di transizione **DELTA**, a partire dall'insieme **q** delle quintuple, lo stato iniziale è quello usato nella prima quintupla.

In[4]:=
```
InitDelta[Q_]:=(
 Clear[DELTA];
 delta[_,_]:={};
 Scan[
  (DELTA[#[[1]],#[[2]]]=
       Drop[#,2])&,
   Q];
 STATO=Q[[1,1]];)
```

In[5]:=
```
q={{A,"0",A,"1",D},
   {A,"1",A,"1",D},
   {A," ",B," ",S}};
```

In[6]:=
```
InitDelta[q];
STATO
?DELTA
```

Out[7]=
```
A
```

Out[8]=
```
DELTA[A, " "] = {B, " ", S}
DELTA[A, "0"] = {A, "1", D}
DELTA[A, "1"] = {A, "1", D}
```

Elaborazione

move[{s,x,d}] effettua una mossa andando nel nuovo stato **s** scrivendo il simbolo **x** e spostandosi a destra se **d==D** e a sinistra altrimenti e rende **True** .

move[{}] rende **False** (questo ci servirà per fermare la macchina).

In[9]:=
```
move[{s_,x_,d_}]:=(
   TAPE[POS]=x;
   POS=POS+If[d==D,1,-1,-1];
   MINPOS=Min[POS,MINPOS];
   MAXPOS=Max[POS,MAXPOS];
   STATO=s1;
   True)
move[{}]=False;
```

save aggiunge alla lista **history** la configurazione corrente.

In[11]:=
```
save:=AppendTo[history,
  {STATO,MINPOS,POS,MAXPOS,TapeToList}];
```

Elaborazione inizializza il nastro e la funzione di transizione ed effettua le mosse finché possibile. Poiché **DELTA** per *default* vale **{}** se una configurazione non ha una quintupla corrispondente **DELTA** rende **{}**, **move** rende **False** e l'esecuzione dell'**while** si ferma.

N.B. **Elaborazione** può andare in ciclo, questo non è un difetto ma una proprietà voluta (per facilitare il *debugging* potremmo fermare la macchina dopo 1000 o 10000 passi, limitandone le possibilità).

In[12]:=
```
Elaborazione[q_,t_]:= (
    InitTape[t];
    InitDelta[q];
    history={};
    save;
    While[move[DELTA[STATO,TAPE[POS]]],
          save]);
```

Dump

Per i nostri scopi didattici è importante tracciare tutta la elaborazione e disegnarne la storia alla fine. N.B. se la macchina va in ciclo non si vede nulla e bisogna seguire altre strade (ve lo lascio come esercizio).

cellc[i,j,t] disegna una cella del nastro con il suo contenuto;

In[13]:=
```
cellc[i_,j_,t_]:={
  Line[{{i, j},{i+1, j},
        {i+1, j+1},{i, j+1},
        {i,j}}],
  Text[t, {i+0.5,j+0.5}]}
```

headc[i,j,s] disegna un rettangolo di un colore assegnato

In[14]:=
```
headc[i_,j_,s_RGBColor]:=
  {s,Rectangle[{i, j},{i+1, j+1}]}
```

line disegna una configurazione (il nastro con il punto di lettura evidenziato con un colore che dipende dallo stato)

In[15]:=
```
line[{s_,mp_,p_,MP_,tape_}]:=
   {headc[p,1.2(-j),color[s]],
    Table[cellc[i,1.2j,
          tape[[i-mp+1]]],{i, mp, MP}]}
```

DumpHistory disegna la storia della elaborazione applicando la funzione **line** ad ogni elemento di **history**. Nel seguito abbiamo usato anche **DumpLargeHistory[ncols]** che disegna su **ncols** colonne (ve lo lascio come esercizio).

In[17]:=
```
DumpHistory:=(
  j=0;
  Show[Graphics[line/@history],
     AspectRatio->-1.2j/(MAXPOS-MINPOS+1)]);
```

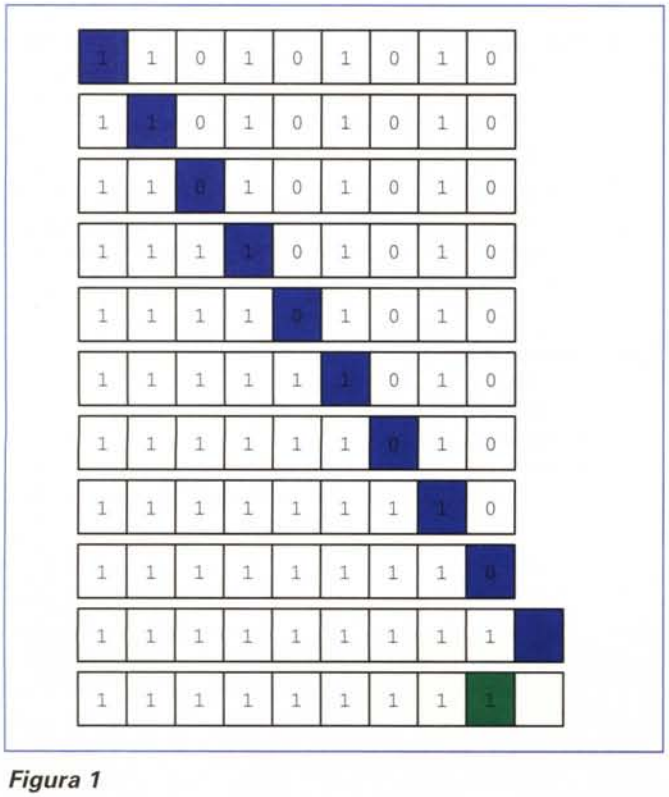

Figura 1

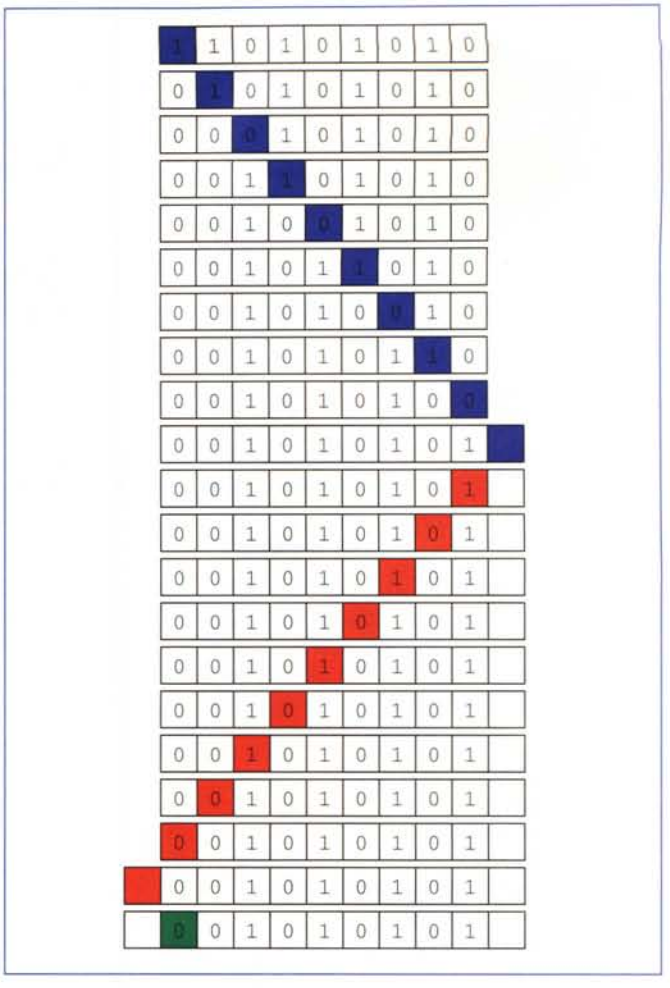

Figura 2

Infine definiamo un colore per ogni stato che utilizzeremo in seguito, tenendoci il grigio come riserva

In[18]:=
```
Needs["Graphics`Colors`"];
color[_]:=Gray;
color[A]=Blue;
color[B]=Red;
color[C]=Cyan;
color[D]=Pink;
color[E]=Magenta;
color[F]=Yellow;
color[G]=Green;
color[H]=Brown;
```

I primi programmi

La prima macchina che simuliamo trasforma gli zeri della stringa di ingresso in uni e poi si ferma. Gli stati sono due: A porta 0 in 1 , 1 in 1 e va a destra, se con A si trova "b" si va in G (lo stato verde) per cui non è definito nulla e quindi ci si ferma.

In[1]:=
```
q={{A,"0",A,"1",D},
   {A,"1",A,"1",D},
   {A," ",G," ",S}};
```

In[2]:=

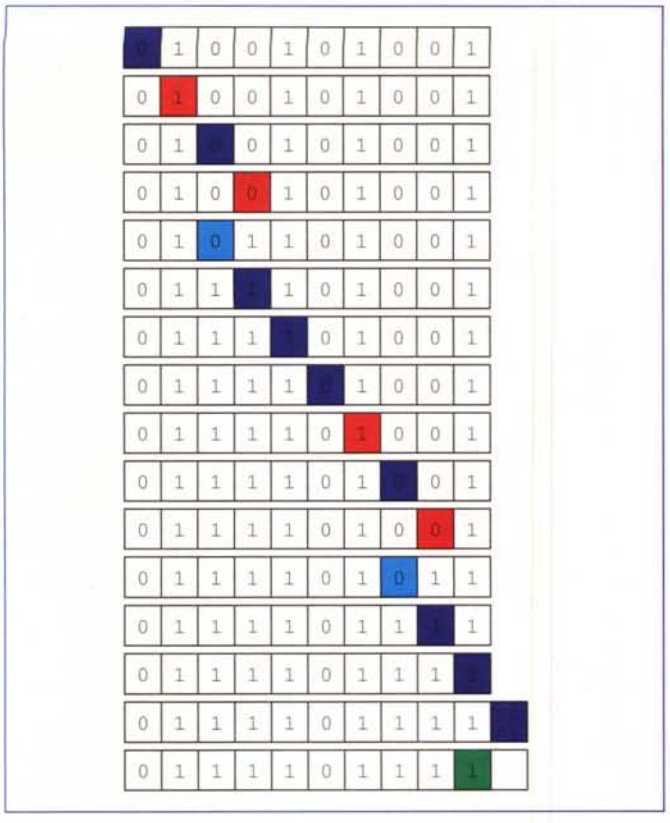

Figura 3

```
Elaborazione[q,"110101010"];
DumpHistory;
```
(vedi figura 1)

La seconda macchina trasforma gli zeri in uni, gli uni in zeri e torna all'inizio della stringa. Lo stato B serve a tornare indietro e anche in questo caso ci fermiamo in uno stato verde.

In[3]:=
```
q={{A,"0",A,"1",D},
   {A,"1",A,"0",D},
   {A," ",B," ",S},
   {B,"0",B,"0",S},
   {B,"1",B,"1",S},
   {B," ",G," ",D}};
```

10 lezioni su *Mathematica*

Nell'ambito dell'insegnamento di **Laboratorio di Matematica Computazionale** del Corso di Laurea in Informatica dell'Università di Pisa verranno tenute 20 ore di corso su *Mathematica*, suddivise in 10 lezioni. Il programma comprende i seguenti argomenti: Introduzione a *Mathematica* - Grafica e programmazione - *Mathematica* come ausilio alla ricerca matematica - Esperimenti di Analisi Numerica - Il calcolo di π - Applicazioni Didattiche - Programmazione avanzata - Grafica avanzata - Elaborazione di testi - Elementi di *Signal Processing*.

La partecipazione è gratuita e aperta a tutti gli interessati. Per informazioni sull'orario e la sede del corso rivolgersi a *Francesco Romani, Dipartimento di Informatica, Corso Italia 40, 56100 Pisa.*

In[4]:=
```
Elaborazione[q,"110101010"];
DumpHistory;
```
(vedi figura 2)

Il terzo esempio di questo tipo trasforma le coppie "00" in "11". Lo stato B viene attivato se si trova uno zero e la sostituzione viene effettuata se se ne trova un secondo.

In[5]:=
```
q={{A,"1",A,"1",D},
   {A,"0",B,"0",D},
   {B,"1",A,"1",D},
   {B,"0",C,"1",S},
   {C,"0",A,"1",D},
   {A," ",G," ",S},
   {B," ",G," ",S}};
```

In[6]:=
```
Elaborazione[q,"0100101001"];
DumpHistory;
```

Un contatore binario

Una stringa sul nastro composta di zeri ed uni può essere considerata un numero binario, vediamo come si scrivono le quintuple che sommano 1. L'idea è di andare in fondo a destra, sostituire "0" in "1" (e avremmo finito) oppure "1" in "0" e allora dobbiamo tornare indietro con un riporto.

In[1]:=
```
q={
(* si va in fondo          *)
   {A,"0",A,"0",D},
   {A,"1",A,"1",D},
   {A," ",B," ",S},
(* si torna senza riporto *)
   {B,"0",C,"1",S},
   {C,"0",C,"0",S},
   {C,"1",C,"1",S},
(* si torna con riporto    *)
   {B,"1",B,"0",S},
   {B," ",C,"1",S},
(* fine                    *)
   {C," ",G," ",D}};
```

In[2]:=
```
Elaborazione[q,"0100111111"];
DumpLargeHistory[2];
```
(vedi figura 4)

Ripetendo più volte la somma di "1" si può implementare un contatore binario ovvero una macchina che trasforma una stringa di uni (un numero unario) nel suo equivalente binario.

La MdT *mangia* un uno alla volta incrementando il contatore e si ferma quando è finita la *pappa*.

In[3]:=
```
q={
(* si va in fondo          *)
   {A,"1",A,"1",D},
   {A," ",B," ",S},
```

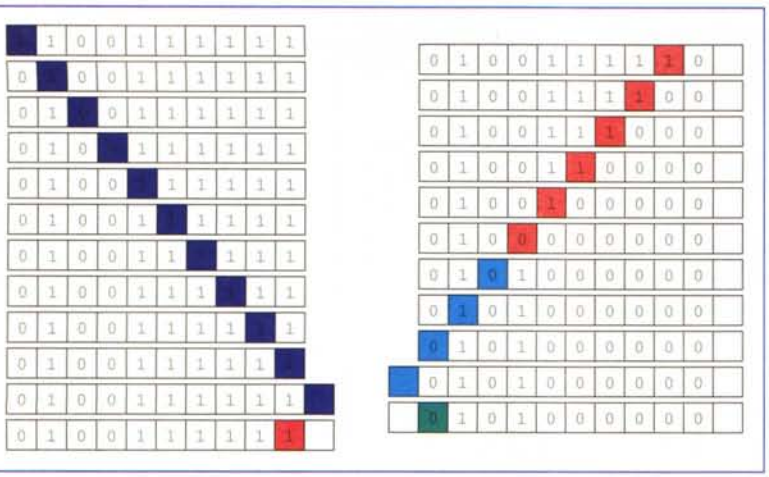

Figura 4

```
(* si toglie un 1          *)
   {B,"1",C," ",S},
(* e si ritorna            *)
   {C,"1",C,"1",S},
   {C," ",D," ",S},
(* si aggiunge 1           *)
   {D,"0",E,"1",D},
   {D,"1",D,"0",S},
   {D," ",E,"1",D},
(* si ricerca la stringa *)
   {E,"1",E,"1",D},
   {E,"0",E,"0",D},
(* e si ricomincia         *)
   {E," ",A," ",D}};
```

In[4]:=

```
Elaborazione[q,"11111111"];
DumpLargeHistory[3];
```

(vedi figura 5)

Riconoscimento di linguaggi

Le Macchine di Turing possono essere usate anche come riconoscitori di linguaggi. Si pone la stringa da esaminare sul nastro di ingresso e la si considera accettata (ovvero facente parte del linguaggio) se e solo se la MdT termina la sua elaborazione in uno stato particolare (per esempio quello verde). Se si termina in uno stato diverso o se non si termina affatto la stringa è rifiutata. Il contenuto del nastro al termine della elaborazione viene trascurato.

Il primo esempio tratta del riconoscimento di stringhe del tipo $a^n b^n$ (per esempio "aaabbb").

La macchina mangia un "a" in cima alla stringa poi va a cercare un "b" in fondo e così via finché il nastro è vuoto.

In[1]:=

```
q={{A,"a",B," ",D},
(* si va in fondo          *)
   {B,"a",B,"a",D},
   {B,"b",B,"b",D},
```

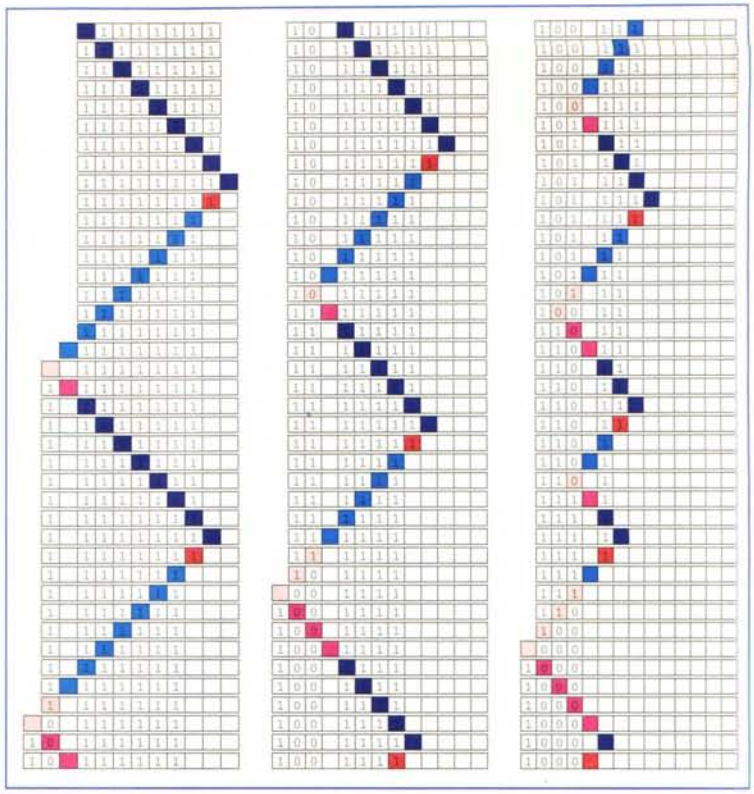

Figura 5

```
   {B," ",C," ",S},
(* si torna indietro       *)
   {C,"b",E," ",S},
   {E,"a",E,"a",S},
   {E,"b",E,"b",S},
(* si ricomincia           *)
   {E," ",A," ",D}
(* se il nastro è vuoto OK *)
   {A," ",G," ",D}};
```

N.B. dimostrare che il programma è corretto è tutt'altro che facile ma non impossibile (salvo errori od omissioni...)

In[2]:=

```
Elaborazione[q,"aaaabbbb"];
DumpLargeHistory[3];
```

(vedi figura 6)

Vediamo anche un esempio di stringa rifiutata.

In[3]:=

```
Elaborazione[q,"aaaabbbbb"];
DumpLargeHistory[3];
```

(vedi figura 7)

Infine riconosciamo una stringa del tipo $a^n b^n c^n$ (per esempio "aaabbbccc"). Per fare ciò si mangiano gli "a" trasformando i "b" in "d" poi si riconosce $d^n c^n$.

In[4]:=

```
q={
(* trattamento di a e b *)
   {A,"a",B," ",D},
   {B,"a",B,"a",D},
   {B,"b",B,"b",D},
   {B,"c",C,"c",S},
   {B,"d",C,"d",S},
```

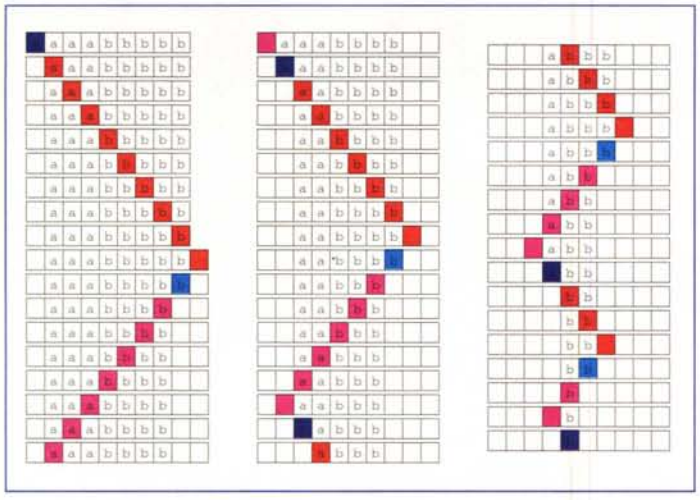

Figura 6

```
   {C,"b",D,"d",S},
   {D,"a",D,"a",S},
   {D,"b",D,"b",S},
   {D," ",A," ",D},
(* trattamento di d e c *)
   {A,"d",E," ",D},
   {E,"d",E,"d",D},
   {E,"c",E,"c",D},
   {E," ",F," ",S},
   {F,"c",H," ",S},
   {H,"d",H,"d",S},
   {H,"c",H,"c",S},
   {H," ",A," ",D},
(* se il nastro è vuoto OK*)
   {A," ",G," ",D}};
```

In[5]:=

```
Elaborazione[q,"aaabbbccc"];
DumpLargeHistory[3];
```

(vedi figura 8)

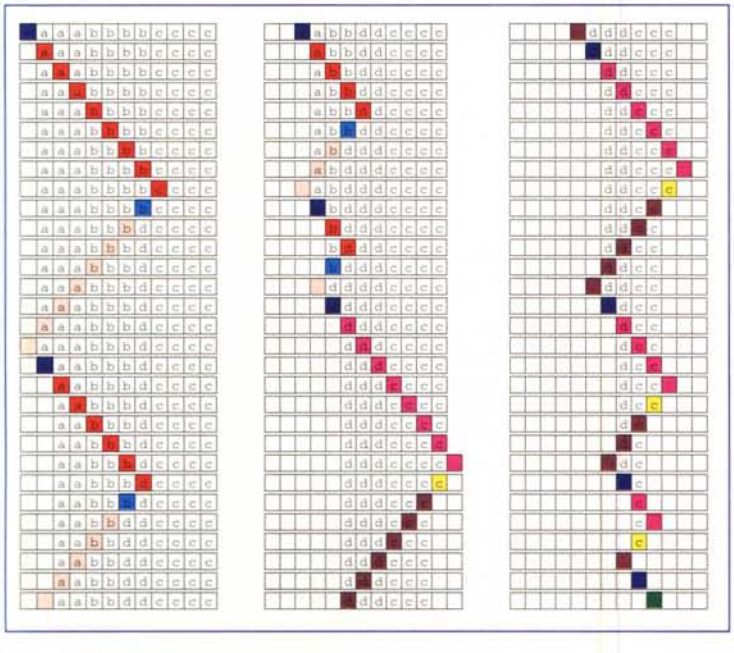

Figura 8

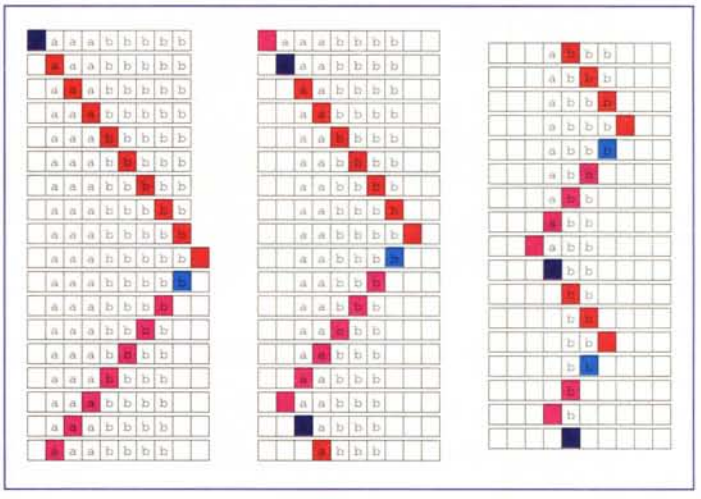

Figura 7

Vediamo anche un esempio di stringa rifiutata.

In[6]:=

```
Elaborazione[q,"aaabbbcccc"];
DumpLargeHistory[3];
```

(vedi figura 9)

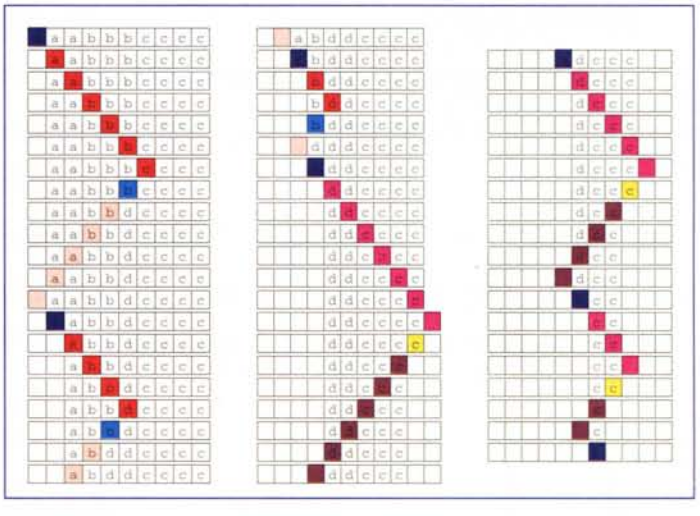

Figura 9

MC

Bibliografia

Roman Maeder, **The *Mathematica* Programmer II,** Academic Press, 1996.

Marvin Minsky, **Computation: Finite and Infinite Machines**, Prentice Hall, 1967.

Stephen Wolfram, **The Mathematica Book, 3rd ed.**, Cambridge University Press, 1996.