La Dieta dei Dati

Una delle aree più eleganti dell'informatica è certamente la Teoria dell'Informazione e della Trasmissione, che oltre alle sue notevoli implicazioni teoriche, pone le basi di molte applicazioni pratiche tra cui la compressione dei dati (la dieta del titolo). Non esiste smanettone del computer che non sia un super-esperto dei programmi di compressione, usati sia per ricevere dati via modem (risparmiando sulla bolletta), che per archiviare (risparmiando dischetti e spazio sull'HardDisk). Questo mese introduciamo in modo elementare (naturalmente con Mathematica*) la codifica e la compressione dei dati*

di Francesco Romani

Codici e codifiche

Tutta l'informatica si basa sul concetto di codifica. Come tutti sanno i calcolatori digitali trattano solo quantità binarie (tipicamente segnali elettrici di due tipi diversi a cui si associano le cifre binarie 0 e 1) e tutti sanno anche che di lavorare con zeri e uni non interessa a nessuno o quasi. La chiave di volta di tutto il sistema è che, con gruppi di cifre binarie, si possono rappresentare numeri, lettere e qualsiasi altra cosa venga in mente (immagini, suoni, realtà virtuale, etc.).
In questa chiacchierata, consideriamo il problema di codificare in binario un simbolo espresso in un alfabeto (per esempio {a,b,c,d}). Il processo avviene in due fasi:
1) si sceglie un insieme di k parole binarie detto **codice**;
2) si assegna in modo univoco una parola del codice ad ogni simbolo dell'alfabeto.
È molto importante che una volta tradotto in binario il simbolo sia ricostruibile senza ambiguità. Deve essere quindi possibile raggruppare agevolmente le cifre binarie del messaggio codificato per formare le parole del codice da cui si risale al messaggio originario. Il modo più semplice per fare ciò è quello di assegnare ad ogni simbolo da codificare un numero fisso di cifre binarie.
Costruiamo la funzione **Codice** che assegna ad una lettera una stringa binaria, e la funzione **Codifica** che converte una stringa su {a,b,c,d} in una binaria, (rIcordiamo che **Characters** rende la lista dei caratteri di una stringa).

In[1]:=
```
Codice["a"]="00";
Codice["b"]="01";
Codice["c"]="10";
Codice["d"]="11";
```
In[2]:=
```
Codifica[s_String]:=
 StringJoin[
   Map[Codice,Characters[s]]]
```
In[3]:=
```
messaggio=Codifica["abcbcd"]
```
Out[3]=
```
000110011011
```
La decodifica è ottenuta attraverso l'uso del Pattern Matching e la ricorsione nelle definizioni di funzioni. I tre *underscore* (_) dopo **resto** significano che quella variabile è composta da 0 o più argomenti.

In[4]:=
```
Decode["0","0",resto___]:=
 StringJoin["a",Decode[resto]];
Decode["0","1",resto___]:=
 StringJoin["b",Decode[resto]];
Decode["1","0",resto___]:=
 StringJoin["c",Decode[resto]];
Decode["1","1",resto___]:=
StringJoin["d",Decode[resto]];
Decode[]:="";
```
Per decodificare una stringa di bit, si chiama **Decode** dando come argomenti tutte le cifre binarie del messaggio

In[5]:=
```
DeCodifica[s_String]:=
 Decode@@Sequence[Characters[s]]
```
N.B. si noti l'uso di **Sequence** per trasformare una lista in una sequenza, più brutalmente si potrebbe dire che vengono tolte le parentesi graffe.

In[6]:=
```
DeCodifica[messaggio]
```
Out[6]=
```
abcbcd
```
Se si vogliono codificare tutte le lettere dell'alfabeto si devono usare parole più lunghe di due bit. Ad esempio vediamo di usare 7 bit del codice ASCII. **ToCharacterCode** trasforma un carattere nella lista dei numeri corrispondenti alle sue possibili rappresentazioni e **IntegerDigits** ne dà le cifre in una base a piacere.

In[7]:=
```
First[ToCharacterCode["z"]]
```
Out[7]=
```
122
```
In[8]:=
```
IntegerDigits[%,2]
```
Out[8]=
```
{1, 1, 1, 1, 0, 1, 0}
```
Otteniamo sette cifre perché 122 ammette una rappresentazione a 7 bit se avessimo provato con il blank avremmo ottenuto meno cifre e con certi caratteri speciali ne avremmo ottenute otto. A noi però interessano le lettere da "a" a "z" che vanno da 97 a 122 e otteniamo sempre 7 bit. La nuova definizione di **Codice** è semplice e facciamo in modo di ricordare i valori calcolati per accelerare l'esecuzione.

In[9]:=
```
Clear[Codice];
Codice[x_]:=Codice[x]=
 StringJoin[
   Map[ToString,
     IntegerDigits[
      First[ToCharacterCode[x]],2]]]
```
La definizione di **Decode** è un po' più involuta: bisogna definire **Decode** per tutti i numeri da 97 a 122 e usare **FromCharacterCode** che trasforma un numero in un caratere.

In[10]:=
```
Clear[Decode];


Decode[]="";
```

```
Do[Decode[
     Sequence@@Map[
      ToString,IntegerDigits[x,2]],
     resto___]:=
    Evaluate[StringJoin[
     FromCharacterCode[x],Decode[resto]]],
{x,97,122}]
```

Una volta definiti **Codice** e **Decode** le funzioni **Codifica** e la **Decodifica** rimangono le stesse.

In[11]:=
```
Testo="precipitevolissimevolmente"
```
Out[11]=
```
precipitevolissimevolmente
```
In[12]:=
```
messaggio=Codifica[Testo]
```
Out[12]=
```
11100001110010110010111000111101001111
  10000110100111101001100101111011011\
  01111110110011010011110011111001111\
  01001110110111001011110110110111111\
  01100110110111001011101110111010011\
  00101
```
In[13]:=
```
StringLength[messaggio]
```
Out[13]=
```
182
```
In[14]:=
```
DeCodifica[messaggio]
```
Out[14]=
```
precipitevolissimevolmente
```

Codici istantanei

Il metodo di codifica con parole della stessa lunghezza è il piu semplice ma non è certo l'unico possibile. In genere si preferisce attribuire codifiche più corte ai simboli di uso più frequente. Esempi comuni di questa tecnica sono l'alfabeto Morse (controllare per credere) e i numeri telefonici. Si attribuiscono numeri brevi ai pubblici servizi (12, 113, 187) e ai centralini dei grandi enti; i numeri delle città più lontane si costruiscono con lunghe sequenze di prefissi (ad esempio il numero 0081-958-4363-3810 potrebbe corrispondere all'interno 3810 di una ditta di Nagasaki).

La caratteristica essenziale richiesta ad un codice con parole di lunghezza diversa è di essere univocamente decifrabile. Anche in questo caso, cioè bisogna poter raggruppare agevolmente le cifre binarie del messagio codificato, per formare le parole del codice da cui risalire al messaggio originario.

Il modo più semplice di fare ciò è imporre che nessuna parola del codice sia prefisso di un'altra. Ciò vale per i numeri telefonici e permette la agevole selezione del numero chiamato instradando la chiamata e lasciando ai selettori remoti il compito di decodificare il resto del codice (nell'esempio precedente dopo il prefisso 0081 internazionale il resto del numero va in Giappone e la centrale giapponese manda a Nagasaki il 4363-3810). Un codice con questa caratteristica si dice **Istantaneo**.

Vediamo come codificare le lettere {a,b,c,d} con un codice istantaneo a lunghezza variabile formato dalle parole {1,01,001,000}.

In[15]:=
```
Clear[Codice];
Codice["a"]="1";
Codice["b"]="01";
Codice["c"]="001";
Codice["d"]="000";
```
Il programma di codifica non cambia.

In[16]:=
```
messaggio=Codifica["ababab"]
```
Out[16]=
```
101101101
```
La definizione ricorsiva di **Decode** che sembrava così complicata, ha però il vantaggio di funzionare anche con i codici a lunghezza variabile (purché istantanei). I lettori possono dimostrare matematicamente questo fatto per induzione (oppure fidarsi di me).

In[17]:=
```
Clear[Decode];

Decode[]:="";

Decode["1",resto___]:=
 StringJoin["a",Decode[resto]];
Decode["0","1",resto___]:=
 StringJoin["b",Decode[resto]];
Decode["0","0","1",resto___]:=
 StringJoin["c",Decode[resto]];
Decode["0","0","0",resto___]:=
 StringJoin["d",Decode[resto]];
```
In[18]:=
```
DeCodifica[messaggio]
```
Out[18]=
```
ababab
```

Codifica ottimale

Se per un codice di k parole è definita una distribuzione di probabilità $p_1, p_2, ..., p_k$ allora si può misurare la **lunghezza media del codice**:

$$r = \sum_{i=1}^{k} |c_i| p_i$$

dove $|c_i|$ è la lunghezza della parola associata al simbolo di probabilità p_i.È ovvia ora la convenienza di assegnare le parole di codice più corte ai simboli più probabili, ma la soluzione ottimale al problema della codifica è tutt'altro che evidente.

Il problema può essere inquadrato in una teoria di fondamentale importanza per l'informatica sia teorica che applicata e viene risolto in modo molto elegante ma che non possamo trattare in dettaglio in questa sede. Esiste un interessante algoritmo, dovuto a Huffman, che, data un insieme di simboli con una assegnata distribuzione di probabilità, permette di trovare una scelta delle parole di codice ottimale da punto di vista della lunghezza media.

Codifica di Huffman È data una distribuzione di probabilità $\{p_1, p_2, ..., p_k\}$ e si suppone $p_1 \le p_2 \le ... \le p_k$. Le due parole più lunghe del codice che verrà generato saranno associate a p_1 e p_2. Usiamo un bit per distinguere tra i due eventi e consideriamo l'evento "accade p_1 o p_2". Tale evento ha probabilità p_1+p_2 e e l'insieme $\{p_1+p_2, p_3, ..., p_k\}$ è ancora una distribuzione di k-1 probabilità. In pratica si considerano i k eventi elementari come k alberi formati dalla sola radice. Ogni volta che i due eventi meno probabili vengono raggruppati, l'evento risultante è un albero formato da un nodo e dai due sottoalberi relativi agli eventi che vengono raggruppati. Iterando il procedimento si ottiene un unico albero binario a cui è associato un codice istantaneo completo. Si può dimo-

strare che tale codice ha una lunghezza media minore di:

$$-\sum_{i=1}^{k} p_i \lceil \log_2 p_i \rceil$$

e che, inoltre, non è possibile trovare un codice con lunghezza media inferiore.
Un'efficiente implementazione di questo algoritmo si può realizzare (in C o in Pascal) utilizzando una coda a priorità realizzata con un heap di alberi binari
In *Mathematica* si può implementare una coda a priorità in modo non eccessivamente efficiente ma molto semplice. La coda è una lista di liste tenute in ordine di priorità crescente. **OutQueue** rende la coppia formata dal primo elemento della lista e dal resto della medesima. **InQueue** inserisce un elemento nella lista mantenendola in ordine crescente.

In[1]:=
```
OutQueue[Queue_]:=
 {First[Queue],Rest[Queue]};
InQueue[Queue_,Item_]:=
 Union[Queue,{Item}];
```
Prepariamo una lista dei simboli da codificare con anteposte le loro probabilità in ordine crescente.

In[2]:=
```
ff={{0.1,"a"},
    {0.1,"b"},
    {0.2,"c"},
    {0.6,"d"}};
```
Costruiamo la coda iniziale inserendo davanti ai simboli la stringa vuota.

In[3]:=
```
Q=Map[{#[[1]],{"",#[[2]]}}&,ff];
InputForm[Q]
```
Out[3]=
```
{{0.1, {"", "a"}},
 {0.1, {"", "b"}},
 {0.2, {"", "c"}},
 {0.6, {"", "d"}}}
```
Le funzioni **add0** e **add1** prepongono rispettivamente uno "0" e un "1" al codice.

In[4]:=
```
add0[a_]:={StringJoin["0",a[[1]]],
           a[[2]]};
add1[a_]:={StringJoin["1",a[[1]]],
           a[[2]]};
```
La funzione **join** unisce due liste di simboli distinguendo la prima con uno 0 e la seconda con un 1 e sommando le relative probabilità..

In[5]:=
```
join[a_,b_]:= {a[[1]]+b[[1]],
  Union[Map[add0,a[[2]]],
        Map[add1,b[[2]]]]}
```
Il programma **huffmann** estrae i due elementi con probabilità più piccola li unisce e rimette dentro il risultato proseguendo finché la coda contiene un solo elemento

In[6]:=
```
huffmann:=
 Do[{a,Q}=OutQueue[Q];
    {b,Q}=OutQueue[Q];
     Q=InQueue[Q,join[a,b]],
   {Length[Q]-1}]
```
Eseguiamo l'operazione e vediamo il risultato.

In[7]:=
```
huffmann
```
In[8]:=
```
Q
```
Out[8]=
```
{{1., {{000, a}, {001, b}, {01, c}, {1, d}}}}
```
Estraiamo la lista dei codici

In[9]:=
```
cod=Sort[Transpose[Reverse[
  Transpose[Q[[1,2]]]]]]
```
Out[9]=
```
{{a, 1}, {b, 001}, {c, 01}, {d, 000}}
```
e organizziamo nello stesso modo la lista delle probabilità.

In[10]:=
```
prob=Sort[Transpose[Reverse[Transpose[ff]]]]
```
Out[10]=
```
{{a, 0.6}, {b, 0.1}, {c, 0.2}, {d, 0.1}}
```
Con un prodotto scalare si calcola la lunghezza media.

In[11]:=
```
Map[StringLength[#[[2]]]&,cod].
 Map[#[[2]]&,prob]
```
Out[11]=
```
1.6
```
Come si vede si è scesi a 1.6 rispetto al valore 2 che avremmo avuto con una codifica a lunghezza fissa su 2 bit.

Codifica per l'italiano

Ora siamo pronti ad applicare questa tecnica per codificare le parole italiane. Come primo passo bisogna procurarsi una lista delle probabilità di occorrenza delle singole lettere. L'unica strada pratica è quella di contare le frequenze delle occorrenze su un campione di testo (possibilmente molto vasto e articolato). Tanto per fare un esempio prendiamo la forma ASCII del primo articolo che ho scritto per MC (n 125 gennaio 1993), lo stesso file usato nel n. 130 per contare le fequenze delle parole). Leggiamo la lista delle parole e la trasformiamo in una lista di caratteri.

In[1]:=
```
lista=ReadList["MC1.txt",String];
Short[lista]
```
Out[1]=
```
{Mathematica, Un, <<1830>>, costosa}
```
Ricordo che **Short[...]** scrive una lunga espressione in un sola riga indicando tra parentesi doppie angolate << >> il numero di elementi tralasciati.
Selezionamo solo le lettere e le convertiamo in minuscole

In[2]:=
```
lc=Flatten[Map[Characters,lista]];
ll=Map[ToLowerCase,Select[lc,LetterQ]];
Short[ll]
```
Out[2]=
```
{m,a,t,h,e,m,a,t,i,<<9950>>,t,o,s,a}
```
Il programma che conta le frequenze delle lettere è molto simile a quello che contava la frequenza delle parole visto nel n.130

In[3]:=
```
Contalettere[ll_]:=Module[{lettere,lungh},
   lettere=Union[ll];
   lungh=Length[ll];
   Sort[Transpose[
    {Map[N[Count[ll,#]/lungh]&,lettere],
     lettere}]];
```
Il risultato è una lista di coppie, {frequenza relativa, lettera}.

In[4]:=
```
ff=Contalettere[lc]
```
Out[4]=
```
{{0.000401485, x}, {0.000501857, y},
 {0.00100371, k}, {0.00190706, w},
 {0.0028104, q}, {0.00993677, b},
 {0.0104386, v}, {0.0111412, h},
```

```
 {0.0123457, z}, {0.0128475, f},
 {0.0174646, g}, {0.0311151, u},
 {0.0328214, d}, {0.0365352, p},
 {0.0411523, m}, {0.0443641, c},
 {0.0537991, s}, {0.0552043, l},
 {0.0612265, r}, {0.0662451, n},
 {0.0734718, t}, {0.0818027, o},
 {0.0982636, a}, {0.118237, e},
 {0.124962, i}}
```

Applichiamo l'agoritmo di Huffmann e otteniamo il codice.

In[5]:=

```
Q=Map[{#[[1]],{"",#[[2]]}}&,ff];
huffmann
```

In[6]:=

```
cod=Sort[Transpose[
 Reverse[Transpose[Q[[1,2]]]]]]
```

Out[6]=

```
{{a, 000},
 {b, 1100001},
 {c, 11111},
 {d, 10101},
 {e, 011},
 {f, 001011},
 {g, 110001},
 {h, 001001},
 {i, 100},
 {k, 1100000111},
 {l, 0100},
 {m, 11110},
 {n, 1011},
 {o, 1110},
 {p, 11001},
 {q, 11000000},
 {r, 0101},
 {s, 0011},
 {t, 1101},
 {u, 10100},
 {v, 001000},
 {w, 110000010},
 {x, 11000001100},
 {y, 11000001101},
 {z, 001010}}
```

Organizziamo nello stesso modo la lista delle probabilità.

In[7]:=

```
prob=Sort[Transpose[Reverse[Transpose[ff]]]]
```

Out[7]=

```
{{a, 0.0982636}, {b, 0.00993677},
 {c, 0.0443641}, {d, 0.0328214},
 {e, 0.118237}, {f, 0.0128475},
 {g, 0.0174646}, {h, 0.0111412},
 {i, 0.124962}, {k, 0.00100371},
 {l, 0.0552043}, {m, 0.0411523},
 {n, 0.0662451}, {o, 0.0818027},
 {p, 0.0365352}, {q, 0.0028104},
 {r, 0.0612265}, {s, 0.0537991},
 {t, 0.0734718}, {u, 0.0311151},
 {v, 0.0104386}, {w, 0.00190706},
 {x, 0.000401485}, {y, 0.000501857},
 {z, 0.0123457}}
```

E calcoliamo la lunghezza media.

In[8]:=

```
Map[StringLength[#[[2]]]&,cod].
 Map[#[[2]]&,prob]
```

Out[8]=

```
4.03593
```

Rispetto ai 5 caratteri per lettera di una codifica a lunghezza fissa su 5 bit abbiamo risparmiato circa un bit per lettera.

Creiamo la funzioni **Codice**

In[9]:=

```
Clear[Codice];
 Scan[SetDelayed[Codice[#[[2]]],
      Evaluate[#[[1]]]]&,Q[[1,2]]]
```

In[10]:=

```
?Codice
```

```
Global`Codice
Codice["a"] := "000"
Codice["b"] := "1100001"
Codice["c"] := "11111"
Codice["d"] := "10101"
Codice["e"] := "011"
...
Codice["w"] := "110000010"
Codice["x"] := "11000001100"
Codice["y"] := "11000001101"
Codice["z"] := "001010"
```

E codifichiamo "precipitevolissimevolmente"

In[11]:=

```
messaggio=Codifica[
 "precipitevolissimevolmente"]
```

Out[11]=

```
1100101010111111111001100110\
0110101100100011100100100000\
1100111001111001100100011110\
0100111100111011111011
```

In[12]:=

```
StringLength[%]
```

Out[12]=

```
104
```

Abiamo usato 104 bit contro i 182 del codice ASCII a 7 bit. La decodifica procede nello stesso modo

In[13]:=

```
Clear[Decode];
Decode[]="";
Scan[SetDelayed[
 Decode[Sequence@@
   Characters[#[[1]]],resto___],
  Evaluate[StringJoin[#[[2]],
     Decode[resto]]]]&,Q[[1,2]]]
```

In[14]:=

```
?Decode
```

```
Global`Decode
Decode[]=""
Decode["0","0","0",resto___]:=
 StringJoin["a",Decode[resto]]
Decode["0","0","1","0","0","0",resto___]:=
 StringJoin["v",Decode[resto]]
...
Decode["1","1","1","1","0",resto___]:=
 StringJoin["m",Decode[resto]]
Decode["1","1","1","1","1",resto___]:=
 StringJoin["c",Decode[resto]]
```

In[15]:=

```
DeCodifica[messaggio]
```

Out[15]=

```
precipitevolissimevolmente
```

Come si può facilmente immaginare, il problema principale di questo metodo di compressione è che le frequenze di occorrenza dei caratteri sono spesso sconosciute o ben lontane da quello che la teoria prevede. In una prossima puntata vedremo altri metodi, più adatti ad un uso pratico, che vengono effettivamente implementati nei programmi di compressione.

MC

Francesco Romani è raggiungibile tramite Internet all'indirizzo romani@di.unipi.it

CYBERTRACKS™
RECORDS

VIRTUAL AUDIO PROJECT

Music for Virtual application - HI-TECH Electronic Music
Virtual Reality - Multimedia

Il futuro della musica elettronica è già realtà.

Cybertracks Records

La prima etichetta discografica italiana di Musica Elettronica, sonorizzazioni per Realta' Virtuali e Applicazioni Multimediali.

Virtual Audio Project

Il primo team internazionale che riunisce i migliori compositori elettronici, sperimentatori e programmatori di computers per realizzare innovativi e stimolanti ambienti sonori utilizzabili anche per applicazioni multimediali e applicazioni immersive di Realta' Virtuale.

CD Audio

Ogni bimestre Cybertracks Records propone le inedite opere del Virtual Audio Project su Audio CD, per offrire a tutti gli appassionati di musica elettronica, di realta' virtuale e di tecnologia, un esclusivo prodotto ascoltabile con un normale lettore CD, senza ricorrere necessariamente all'uso di un computer. Ogni CD contiene oltre 60 minuti di brani inediti.

Il Book

Ogni CD viene offerto in una speciale confezione che comprende un Book di 36 pagine in formato A4 sul quale sono presenti le descrizioni dei temi relativi ad ogni traccia del CD, numerosi articoli ed interviste sulla Realta' Virtuale, le applicazioni multimediali, le tecnologie informatiche e musicali.

Le applicazioni:

Gli esperti compositori e programmatori che animano questo esclusivo progetto utilizzano tutta la tecnologia applicabile al settore musicale per realizzare inedite situazioni con ricercate ed innovative sonorita'. Gli ambienti sonori tridimensionali saranno una piacevole occasione di ascolto per vivere stimolanti immersioni audio-guidate con il solo uso del lettore CD; le descrizioni delle singole tracce, presenti sul Book, saranno una valida introduzione per poter vivere realmente le situazioni sonore scelte. Non a caso CYBERTRACKS e' fornitore ufficiale di sonorizzazioni per le applicazioni di Realta' Virtuale realizzate da Virtualitalia Industrie. Non a caso CYBERTRACKS e' presente in tutte le piu' importanti manifestazioni e mostre di HI-FI, musica, computers e Realta' Virtuale. I CD Audio sono stati realizzati per essere facilmente utilizzati anche nella sonorizzazione professionale delle vostre applicazioni multimediali e fornire le ideali colonne sonore per filmati e sequenze in computer grafica. Gli esclusivi prodotti del Virtual Audio Project sono offerti da Cybertracks Records, etichetta discografica del gruppo Novaera, solo su CD Audio.

Il team internazionale di compositori e' formato da J.L.A. Spyris, Andrew Rhodes, Riccardo Ricci, Aguri Kayajama, Phil M.Trask, Aldo Polverari, Christian Weber, Realogic, Richard Deelay e Dioxide.

GIA' DISPONIBILE		"MIND CONTACT" (Cod.IVARC0001) • ISSUE 01 La compilation che meglio rappresenta le creazioni del Virtual Audio Project (14 Tracce). Lit. 24.900
GIA' DISPONIBILE		"ACCELERATION" (Cod.IVARC0002) • ISSUE 02 Velocità, fughe, cadute libere e inseguimenti. Oltre 60 min. di rischiose esperienze ad alta velocità. Lit. 24.900
SET 94		"INTO BRAIN" (Cod.IVARC0003) • ISSUE 03 Affascinanti esperienze per vivere le fasi della nascita, sviluppo e crescita del pensiero. Lit. 24.900
NOV 94		"EXPLORATION" (Cod.IVARC0013) • ISSUE 04 Suggestivi e realistici ambienti spaziali per avventurose immersioni e solitarie esplorazioni. Lit. 24.900
GEN 95		"RELAX DISC" (Cod.IVARC0004) • ISSUE 05 Diverse situazioni per immergersi in tranquilli e rilassanti ambienti sonori, il vostro momento di calma per sfuggire allo stress. Lit. 24.900
MAR 95		"EGO" (Cod.IVARC0015) • ISSUE 06 Coscienza, paure e sensazioni. Differenti esperienze per avvicinarsi al proprio io. Lit. 24.900
MAG 95		"IMPACT" (Cod.IVARC0016) • ISSUE 07 Scontri, violenti impatti e rischiose situazioni per diverse reazioni. Lit. 24.900

NOVAERA

CYBERTRACKS RECORDS is a division of NOVAERA ED. s.a.s.
C.P. 88 - 61100 PESARO
Tel./Fax. 0721- 202.700

NOVAERA

> Si cercano rivenditori per interessante programma conto-vendita. <

CYBERTRACKS Records e' presente a:

SIM HI-FI — Milano - 15/19 Set 94

Realtà Virtuale Expò — Logica e Segno, per capire la Magia del 21° Secolo. — Roma - 29 Set / 2 Ott 94

CD + BOOK
TRACCE INEDITE
AUDIO CD DDD

ACQUISTO SINGOLO
Acquisto di uno o piu' numeri completi di relativo Virtual Book a L. 24.900 ciascuno + 2.800 per spese di spedizione.

ABBONAMENTO ANNUALE (6 NUMERI).
- Sconto 25 % sul prezzo dei 6 numeri.
- Spedizioni postali gratuite.
- Ulteriore sconto 10 % sull'acquisto delle successive produzioni non comprese nell'abbonamento.

Compilare in stampatello e barrare le caselle scelte.

Spedire a: / Send to: **NOVAERA s.a.s. - C.P. 88 - 61100 PESARO - ITALY**

A

Titolo	Quantita'	Prezzo unitario	Prezzo Totale
"MIND CONTACT" - Issue 01		L. 24.900	
"ACCELERATION" - Issue 02		L. 24.900	
		L. 24.900	
		L. 24.900	

Contributo fisso per spese di spedizione (Italia ed Europa) ..... L. 2800

TOTALE DELL'ORDINE ..... L.

B

ABBONAMENTO ANNUALE : n.6 CD + n.6 Books
Sconto 25% - Spese postali gratuite - Ulteriore sconto 10% su produzioni fuori abbonamento
Italia ed Europa : L. 112.000 Tutto compreso.
The world : L. 112.000 + L.10.000 per contributo spese di spedizione.

L'abbonamento dovra' partire dal CD n. ISSUE 0 ..... L. 112000

NAME
VIA / ADDRESS
CITTA' / CITY — PROV/STATE
CAP / ZIP — TEL. / PHONE

Modalita' di pagamento:

- ☐ Allego assegno/cheque n° — di L.
  della Banca — intestato a Novaera Ed. s.a.s.
- ☐ Allego ricevuta del versamento sul c/c postale n° 10223618 intestato a Novaera Ed. s.a.s.
- ☐ Ricevero' il materiale in contrassegno postale
- ☐ Carta di Credito / Credit Card ☐ Eurocard ☐ ☐ VISA

Addebitate l'importo sulla mia Carta di Credito: (compilare il n.o per intero completo anche delle ultime 4 cifre)

N.o

Data di scadenza

Gli ordini con Carta di Credito, privi di firma, non sono validi. Controllate che il n.o della Carta sia corretto! Grazie.

Data: FIRMA

Music for Virtual application
HI-TECH Electronic Music
Virtual Reality
Multimedia

Il futuro della musica elettronica e' gia' realta'.

CYBERTRACKS™ RECORDS
VIRTUAL AUDIO PROJECT