AMIGA È SALVA?

GRUPPO EDITORIALE JACKSON

MAGAZINE AMIGA

ANNO 7
OTTOBRE
1994

L. 14.000
Frs. 14,00

IL MENSILE JACKSON PER GLI UTENTI DI AMIGA

IN ESCLUSIVA
il quinto disco del kit
Commodore
per programmatori

GRUPPO EDITORIALE JACKSON

RIVISTA UFFICIALMENTE RICONOSCIUTA DA COMMODORE ITALIANA

IN PROVA:
- GVP GFORCE 4000
- TANDEM PER 1200
- DKB 4091
- VLAB PAR
- BC 1208
- NEC CDR 201
- VIDEO CREATOR PER CD32

TRANSACTION:
LE PAGINE DEL PROGRAMMATORE

ON DISK 1:
- ARCHANDLER: ARCHIVI SENZA PROBLEMI
- DB21: UN POTENTE DATABASE
- MONED: MODIFICARE GLI SCHERMI
- MP103: VEDERE ANIMAZIONI MPEG
- MULTISAMPLE: CONVERSIONE CAMPIONI
- BLACKJACK: COME AL CASINÒ
- INOLTRE: SCRMENU, OPTICON, KILL

ON DISK 2:
- COMMODORE 3.1 AMIGA DEVELOPER
- UPDATE DISK 5: SOFTWARE TOOLKIT

SPEDIZIONE IN ABBONAMENTO POSTALE / 50 TAXE PERCUE (TASSA RISCOSSA) MILANO CMP ROSERIO

Compressione dei suoni

Analizziamo l'algoritmo distruttivo di Fibonacci

EMANUELE VIOLA

Comprimere adeguatamente i suoni è sempre stato un obiettivo che in pochi hanno raggiunto. Infatti un campionamento, nella maggior parte dei casi, differisce dai file più comunemente soggetti a compressione: un testo ASCII o una immagine sono costituiti per lo più da sequenze di byte simili fra loro e comunque raramente sfruttano tutti i valori (da 0 a 255) che un byte offre. Per esempio, nel testo che state leggendo, sono stati usati finora solo i valori ASCII relativi all'alfabeto italiano (una frazione dei 256 caratteri ASCII) e quindi tale testo potrebbe essere compresso in modo estremamente efficace. Infatti, la maggior parte degli algoritmi di compressione ha come scopo fondamentale l'esprimere ogni byte con pochi bit: per l'alfabeto basterebbero 5 bit, che assicurano 32 combinazioni; solo facendo corrispondere ognuno di queste a una lettera avremmo già ridotto il file quasi della metà rispetto alla lunghezza originaria: ogni lettera infatti occupa un byte (8 bit) che verrebbe espresso utilizzando solo 5 bit. Per un campionamento non accade la stessa cosa, basta aver osservato almeno una volta i dati che compongono un file sonoro (per esempio con AudioMaster) per rendersi conto di come siano sfruttate, in sequenze diverse, tutti i valori offerti da un byte.

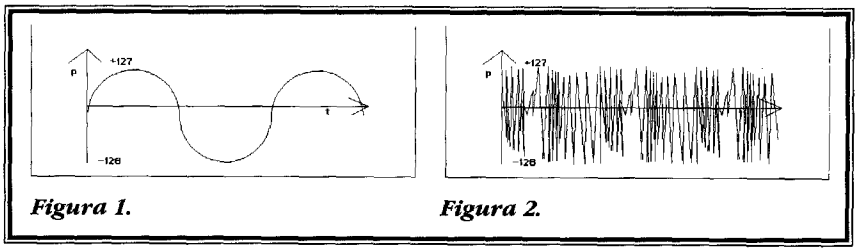

Figura 1. *Figura 2.*

L'algoritmo distruttivo di Fibonacci

Malgrado ciò, l'algoritmo di Fibonacci riesce a portare qualunque file sonoro dalla lunghezza N alla lunghezza N/2+2. Si tratta di un algoritmo distruttivo; ciò significa che una volta eseguita una compressione e poi una decompressione il file ottenuto non sarà più uguale a quello originale, ma risulterà una approssimazione: alcuni byte saranno uguali, altri invece avranno un valore vicino, ma non identico, a quello del file originale. Per questo motivo l'algoritmo NON DEVE essere usato su file come i programmi eseguibili e i testi, che necessitano che ogni bit sia al suo posto.
L'algoritmo sfrutta il fatto che il suono è una forma d'onda e quindi le differenze tra un byte e il successivo dovrebbero essere comprese, la maggior parte delle volte, in un "piccolo" range, esattamente in valori compresi tra +5 e -5. L'algoritmo approssima il file considerando solo le differenze più ricorrenti (quindi non tutte) tra un byte e l'altro (che, come abbiamo visto, sono quelle comprese nel range di cui sopra) e le esprime grazie a nibble (4 bit) che sono la metà di un byte (8 bit), proprio per questo il file viene dimezzato (non contando i 2 byte di header).

Buona e cattiva approssimazione

La figura 1 mostra un file che verrà approssimato bene dal Fibonacci. File di questo tipo sono ottenibili campionando a frequenze molto elevate. La figura 2 mostra un file che verrà approssimato male dal Fibonacci. File di questo tipo sono per lo più rumori sporchi o suoni complessi derivati dalla sovrapposizione di moltissime forme d'onda che, sommandosi, causano valori "molto differenti" tra un byte e l'altro.

Compressione

Chiamiamo il file originale e quello compresso rispettivamente Source e Dest e consideriamo un byte, OldByte, che sarà occupato durante la compressione dal carattere che precede quello che stiamo per analizzare. Consideriamo anche una tabella di 16 valori, espressi in byte, che chiamiamo DiffArray. La tabella conterrà i seguenti valori decimali:

-34 -21 -13 -8 -5 -3 -2 -1 0 1 2 3 5 8 13 21

(1) Scriviamo 0 nel primo byte di Dest.
(2) Copiamo il primo byte di Source nel secondo di Dest e scriviamolo anche in OldByte.
(3) Analizziamo la differenza tra OldByte e il successivo byte di Source, cerchiamo il valore di DiffArray che più gli si avvicina e consideriamo il nibble che rappresenta l'offset di questo valore rispetto all'inizio di DiffArray (cioè 0 per -34, 1 per -21, 15 per 21 e così via). Siamo sicuri di utilizzare al massimo un nibble, poiché la tabella contiene 16 valori perciò l'offset varia da 0=%0000 a 15=%1111.
(4) Scriviamo il nibble del punto 3 nel prossimo nibble di Dest.
(5) Il nuovo valore di OldByte sarà uguale al vecchio valore

```
;*************************************************************
;*                      Emanuele Viola                       *
;*                                                           *
;* Routine in assembly per la decompressione di file con     *
;* algoritmo Fibonacci. Scritto per AmigaMagazine (TransAction)*
;*                                                           *
;* Assemblare con Asm-One (v1.16)                            *
;*************************************************************

    section text,code

_Main:          ;Esempio di chiamata alla routine
      move.l #Source,a0
      move.l #Dest,a1
      move.l #EndSource-Source,d7
      bsr.s _DecrunchFibonacci
      RTS

_DecrunchFibonacci: ;(A0 = Source APTR / A1 = Dest APTR / D7 =
CompressedSize LONG)

    movem.l d0-d5/d7/a0-a2,-(sp)

    moveq #1,d2     ;Tre valori che serviranno
    moveq #$0f,d3   ;per rendere più veloce
    moveq #4,d4     ;la routine.

    lea.l DiffArray(pc),a2

    move.W (a0)+,d0   ;A noi interessa solo il secondo
                      ;byte ma è più veloce di
                      ;addq.w #1,a0 & move.b (A0)+,d0

    subq.l #2,d7      ;Abbiamo saltato i primi due byte
    clr.w d1

.Loop:
                      ;Fase 1 : Primo nibble.
    move.b (a0),d1
    lsr.b d4,d1
    add.w d1,a2
    add.b (a2),d0
    move.b d0,(a1)+
    sub.w d1,a2
                      ;Fase 1 : Secondo nibble.
    move.b (a0)+,d1
    and.b d3,d1
    add.w d1,a2
    add.b (a2),d0
    move.b d0,(a1)+
    sub.w d1,a2
    sub.l d2,d7
    bne.s .Loop

.Exit:
    movem.l d0-d5/d7/a0-a2,-(sp)
    RTS

DiffArray:
    dc.b -34,-21,-13,-8,-5,-3,-2,-1,0,1,2,3,5,8,13,21

;Parte utile per l'esempio di chiamata alla routine.

Source:
    incbin 'ram:ScriviQuiIlNomeDelFileCompressò EndSource:

Dest:
    blk.b (EndSource-Source-2)*2 EndDest:
```

più il valore contenuto nel DiffArray all'offset che abbiamo appena scritto nel nibble al punto 4.
(6) Continuare dal punto 3 finché non finisce il file.

Si noti bene che la prima volta che si passa per il punto 3 si calcola la differenza tra OldByte, che precedentemente era stato posto uguale al primo byte di Source, e il primo byte di Source, quindi fra due valori identici. Perciò il primo nibble da scrivere in Dest sarà $8, che corrisponde all'offset per il valore 0, cioè nessun cambiamento. Questa convenzione è stata adottata per rendere la routine il più generica possibile.

Decompressione

Dato che molti programmi (come AudioMaster) comprimono i dati utilizzando Fibonacci, vogliamo offrire un sorgente in Assembly, essendo importante la velocità, che decomprime secondo quell'algoritmo.
Passiamo quindi a commentare il sorgente Assembly di decompressione. La routine vera e propria inizia a _DecrunchFibonacci. Il resto è solo un esempio di chiamata. La routine necessita in A0 di un puntatore al file compresso, in A1 di un puntatore alla zona di memoria dove si vuole che il file venga decompresso e in D7 la lunghezza del file compresso. Si comincia con il salvataggio dei registri utilizzati, poi se ne inizializzano altri con valori utili per le ottimizzazioni.

Dopo aver caricato in D0 il byte che ci interessa, si entra nel loop, che continuerà finché D7 (inizialmente uguale alla lunghezza del file compresso) non sarà uguale a 0, verrà infatti decrementato di uno, ogni due byte in output, in quanto esprime la lunghezza del file in input in byte a ciascuno dei quali corrispondono due byte in output (1 byte in input = 2 nibble = 2 byte in output).
Il loop è molto semplice: si carica in D1 il prossimo byte, del quale serve solo un nibble (prima fase): perciò si esegue uno shift a destra. Il valore ottenuto viene usato come offset per DiffArray e il byte trovato verrà aggiunto a D0 che andrà in output. La seconda fase è molto simile alla prima, tranne per il fatto che si esegue un:

```
move.b (A0)+,d1
```

così da puntare al successivo byte di Source e un AND perché questa volta interessa il nibble più basso. L'algoritmo non è stato ottimizzato al massimo anche perché potesse risultare più chiara la sua logica di funzionamento.