Heapsort in linguaggio macchina

ovvero, 1000 nomi in ordine alfabetico in 5 secondi

di Bo Arnklit

Nell'opera "The Art of Computer Programming" l'autore, Donald Knuth, afferma che gran parte del tempo dei calcolatori è impiegato nel riordiino alfabetico di dati, o perché la gente fa spesso riordini non necessari, oppure perché vengono adoperate routine poco efficienti. Nella prima metà del terzo volume (Sorting & Searching, 700 pagine), dedicata al problema del sort si comincia con i metodi come il Bubblesort e l'Insertion Sort, che sono semplici sia concettualmente sia come programmazione. Lo svantaggio di questi metodi è che il numero di operazioni è proporzionale al quadrato del numero dei dati da riordinare: così, mentre per pochi dati (10-50) il tempo è breve, passando ad un numero di dati elevato (maggiore di 1000) possono passare delle ore prima che il riordino sia completato, specie se il programma è in BASIC. Sono quindi stati studiati altri metodi come l'Heapsort, il Quicksort e lo Shell Method, per i quali il tempo necessario è proporzionale a N*Log (N) e, quindi, cresce più lentamente con l'aumentare del numero dei dati.

Vi presentiamo, senza stare a vedere, in termini di tempo di esecuzione, i pregi dei vari metodi e senza spiegarne in dettaglio il funzionamento, l'Heapsort tradotto in linguaggio macchina: è così veloce che, con ogni probabilità, non avrete bisogno di altri programmi di sort. Ad esempio, il tempo di riordino per 1000 parole è di circa *5 secondi* e considerando che spesso queste parole (nomi, cognomi e indirizzi) devono essere caricate da disco, facendo parte di un Data Base od altro, il tempo di riordino diventa assolutamente trascurabile rispet-

```
93A8- 20 B1 00 20 E3 DF 85 DC
93B0- 84 DD 20 B1 00 20 E3 DF
93B8- 85 94 84 95 38 E5 DC 85
93C0- AB 98 E5 DD 85 AC A5 AB
93C8- F0 06 C9 01 F0 02 D0 08
93D0- A5 AC D0 04 20 95 D9 60
93D8- 20 70 95 46 42 66 41 18
93E0- A5 41 69 01 85 41 90 02
93E8- E6 42 A5 41 D0 4D A5 42
93F0- D0 49 A6 83 A4 84 20 4F
93F8- 95 A0 00 B1 06 85 AB C8
9400- B1 06 85 AC C8 B1 06 85
9408- AD B1 DC 91 06 88 B1 DC
9410- 91 06 88 B1 DC 91 06 A5
9418- 83 D0 02 C6 84 C6 83 A5
9420- 83 D0 37 A5 84 D0 33 A0
9428- 00 A5 AB 91 DC C8 A5 AC
9430- 91 DC C8 A5 AD 91 DC 20
9438- 95 D9 60 A5 41 D0 02 C6
9440- 42 C6 41 A6 41 A4 42 20
9448- 4F 95 A0 00 B1 06 85 AB
9450- C8 B1 06 85 AC C8 B1 06
9458- 85 AD A5 41 85 71 A5 42
9460- 85 72 A5 71 85 3C A5 72
9468- 85 3D D0 0A A5 71 D0 06
9470- A9 01 85 71 D0 04 06 71
9478- 26 72 A5 72 C5 84 F0 04
9480- 90 0A B0 73 A5 71 C5 83
9488- F0 4A B0 6B A6 71 A4 72
9490- 20 4F 95 18 A5 06 69 03
9498- 85 08 A5 07 69 00 85 09
94A0- 20 3E 95 A0 FF C8 C4 A5
94A8- F0 0C C4 A8 F0 30 B1 A6
94B0- D1 A9 F0 F1 B0 28 A5 08
94B8- 85 06 A5 09 85 07 A5 A8
94C0- 85 A5 A5 A9 85 A6 A5 AA
94C8- 85 A7 E6 71 A5 71 D0 0E
94D0- E6 72 50 0A A6 71 A4 72
94D8- 20 4F 95 20 B6 95 A0 FF
94E0- C8 C4 A5 F0 12 C4 AB F0
94E8- 08 B1 A6 D1 AC F0 F1 90
94F0- 06 A6 3C A4 3D 50 1A A6
94F8- 3C A4 3D 20 4F 95 A0 00
9500- A5 AB 91 06 C8 A5 AC 91
9508- 06 C8 A5 AD 91 06 4C EA
9510- 93 86 08 84 09 06 08 26
9518- 09 18 8A 65 08 85 08 98
9520- 65 09 85 09 18 A5 08 65
9528- DC 85 08 A5 09 65 DD 85
9530- 09 A0 03 88 B1 06 91 08
9538- 98 D0 F8 4C 62 94 A0 03
9540- 88 B1 06 99 A5 00 B1 08
9548- 99 A8 00 98 D0 F2 60 86
9550- 06 84 07 06 06 26 07 18
9558- 8A 65 06 85 06 98 65 07
9560- 85 07 18 A5 06 65 DC 85
9568- 06 A5 07 65 DD 85 07 60
9570- A5 AB 85 9D A5 AC 85 9E
9578- A9 00 85 9F 85 A0 A9 03
9580- 85 A1 A9 00 85 A2 20 96
9588- 95 A5 9D 85 83 85 41 A5
9590- 9E 85 84 85 42 60 A0 10
9598- 06 9D 26 9E 26 9F 26 A0
95A0- 38 A5 9F E5 A1 AA A5 A0
95A8- E5 A2 90 06 86 9F 85 A0
95B0- E6 9D 88 D0 E3 60 A0 00
95B8- B1 06 85 A5 C8 B1 06 85
95C0- A6 C8 B1 06 85 A7 60 69
```

Figura 1

to al tempo di caricamento dei dati.
Il programma, lungo poco più di 1/2 K, è stato assemblato a partire da $93A8 che corrisponde a 37800 in decimale, in modo che può essere inserito nella parte alta della memoria RAM subito sotto il DOS e può essere protetto spostando l'HIMEM a 37800. Per ottimizzare al massimo il programma in termini di velocità i Loop interni sono stati scritti per esteso, cioè, per quanto possibile, senza l'uso di Subroutine, considerando che ogni salto ad una Subroutine porta via circa 12 microsecondi. Una conseguenza, però, è che il programma diventa un po' più lungo. L'eccezionale velocità è anche dovuta, in parte, al modo in cui l'Apple gestisce le stringhe. Ogni stringa è caratterizzata da tre byte: uno per la lunghezza (ecco perché la lunghezza massima delle stringhe è limitata a 256 caratteri) e due byte per l'indirizzo del primo carattere della stringa. I caratteri che compongono la stringa possono essere in qualsiasi parte libera della memoria RAM, normalmente iniziando da HIMEM andando giù, mentre i tre byte che caratterizzano la stringa si trovano in una apposita tabella. Quando una stringa viene modificata, i nuovi caratteri non vengono sostituiti ai vecchi, ma sono immagazzinati in una nuova area di memoria libera, mentre i due byte dell'indirizzo ed il nuovo byte della lunghezza vengono modificati per puntare sulla nuova stringa. Quindi, durante il riordino, per scambiare due stringhe non è necessario scambiare i caratteri delle stringhe stesse, ma è sufficiente scambiare i puntatori ed i byte della lunghezza. In questa maniera il tempo di riordino è quasi indipendente dalla lunghezza delle stringhe.
Il codice macchina riprodotto nella figura 1 viene inserito in memoria come al solito dal monitor: CALL-151

*93A8 : 20 B1 00 20 E3 DF .... etc.

Alla fine dell'inserimento si salva su disco scrivendo:

BSAVE HSORT.OBJ,A$93A8,L$230 (RET)

Per verificare che non vi siano errori di inserimento, potete far girare il programmino riportato nella figura 2, che come risultato deve dare il numero 66351, somma di tutti i byte del programma. Questo procedimento non garantisce che i dati siano giusti (potrebbero annullarsi due errori complementari), ma se il numero è diverso da 66351 c'è sicuramente un errore. Attenzione a non confondere la lettera B con il numero 8.

L'uso dell'Heapsort è estremamente semplice. Supponiamo di avere un Array di stringhe chiamato per esempio A$ contenente 10 elementi (A$(0), A$(1),....A$(9)). Basta inserire la seguente riga nel programma di APPLESOFT:

10 CALL 37800:A$(0),A$(9)

Se volessimo riordinare solo gli elementi da 3 a 8 ad esempio, dovremmo scrivere:

10 CALL 37800:A$(3),A$(8)

È possibile anche riordinare le stringhe cominciando non dal primo carattere, ma da uno successivo: è utile per esempio in un Data Base a campi fissi in cui il cognome cominci al decimo carattere. Per ottenere ciò bisogna eseguire due POKE prima del CALL al programma di sort:

POKE 38052,N-2 : POKE 38111,N-2

dove N è la posizione del primo carattere

```
10  FOR I = 37800 TO 38342
20 X = X +  PEEK (I): NEXT
30  PRINT "SOMMA= ";X
```

Figura 2

```
10  REM GENERATORE DI PAROLE
20  REM     --28/10/1981--
30  REM       BO ARNKLIT
40  REM
50  PRINT  CHR$ (4);"MONI,O"
60  PRINT "OPENRANDOM,L10"
70  FOR J = 0 TO 2000: FOR I = 0 TO 4 + 5 *  RND (1)
80 A$ = A$ +  CHR$ ( RND (1) * 26 + 65): NEXT
90  PRINT "WRITERANDOM,R";J
100  PRINT A$:A$ = "": NEXT
110  PRINT "CLOSERANDOM"
```

Figura 3

```
10  REM  HEAPSORT  DEMO
20  REM  --28/10/1981--
30  REM    BO ARNKLIT
40  REM
50  HIMEM: 37800
60  PRINT  CHR$ (4);"BLOADHSORT.OBJ,A37800"
70  HOME
80  VTAB 10
90  INPUT "NUMERO DI PAROLE   ?";IMAX
100 IMAX = IMAX - 1
110  DIM A$(IMAX)
120  PRINT "OPENRANDOM,L10"
130  FOR J = 0 TO IMAX
140  PRINT "READRANDOM,R";J
150  INPUT A$(J)
160  NEXT
170  PRINT "CLOSERANDOM"
180  FOR J = 0 TO IMAX: PRINT J; TAB( 6);A$(J): NEXT
190  PRINT  CHR$ (7)
200  CALL 37800:A$(0),A$(IMAX)
210  PRINT  CHR$ (7)
220  FOR J = 0 TO IMAX: PRINT J; TAB( 6);A$(J): NEXT
230  GET K$: IF K$ (  ) "Q" THEN  RUN 70
```

Figura 4

preso in considerazione per il riordino.
Per poter meglio apprezzare le straordinarie capacità del nostro Heapsort abbiamo realizzato un DEMO costituito dai due programmini riprodotti nelle figure 3 e 4. Il primo genera 2000 parole casuali con lunghezza compresa tra 4 e 9 caratteri che vengono salvati in un file, denominato Random, su disco in modo che le parole da riordinare siano sempre le stesse, caratteristica fondamentale se si devono fare dei confronti con altri programmi di sort. Il programma Heapsort Demo (f. 4) carica in memoria dal file Random il numero di parole specificato dall'operatore, le visualizza, suona il BEEP, fa il riordino, suona un altro BEEP e visualizza la lista riordinata. Quindi il tempo tra i due BEEP e il tempo impiegato per il riordino. Per un numero di parole inferiore a 100 il riordino è praticamente istantaneo, mentre per 1000 parole ci vogliono circa 5 secondi e per 2000 parole poco più di 10 secondi. Provare per credere!

Sei programmi da una riga

```
10  REM 6 PROGRAMMI DA UNA RIGA
20  REM COPYRIGHT 1981
30  REM BO ARNKLIT
40  REM
50  REM
60  REM HEX)DEC
70  REM
80 A = 0:S$ = "0123456789ABCDEF": INPUT A$: FOR I = 1 TO
     LEN (A$):W$ =  MID$ (A$,I,1):N = 0: FOR J = 1 TO
    16:N = N + (W$ =  MID$ (S$,J,1)) * (J - 1): NEXT
    :A = A + N * 16 ↑ ( LEN (A$) - I): NEXT : PRINT A
    : GOTO 80
90  REM
100  REM
110  REM HEX)BIN
120  REM
130 A = 0:S$ = "0123456789ABCDEF": INPUT A$: FOR I = 1
     TO  LEN (A$):W$ =  MID$ (A$,I,1):N = 0: FOR J =
    1 TO 16:N = N + (W$ =  MID$ (S$,J,1)) * (J - 1):NEXT
    :A = A + N * 16 ↑ ( LEN (A$) - I): NEXT : FOR I =
     INT ( LOG (A) /  LOG (2)) TO 0 STEP  - 1:N =  INT
    (A / 2 ↑ I): PRINT N;:A = A - N * 2 ↑ I: NEXT: PRINT
    : GOTO 130
140  REM
150  REM
160  REM DEC)HEX
170  REM
180 A$ = "0123456789ABCDEF": INPUT A: FOR I =  INT ( LOG
    (A) /  LOG (16)) TO 0 STEP  - 1:B =  INT (A / 16 ↑
    I): PRINT  MID$ (A$,B + 1,1);:A = A - (16 ↑ I) *
    B: NEXT : PRINT : GOTO 180
190  REM
200  REM
210  REM DEC)BIN
220  REM
230  INPUT A: FOR I =  INT ( LOG (A) /  LOG (2))TO 0 STEP
     - 1:N =  INT (A / 2 ↑ I): PRINT N;:A = A - N * 2
     ↑ I: NEXT : PRINT : GOTO 230
240  REM
250  REM
260  REM BIN)HEX
270  REM
280  INPUT A$:A = 0: FOR I = 1 TO  LEN (A$):A = A +  VAL
    ( MID$ (A$,I,1)) * 2 ↑ ( LEN (A$) - I): NEXT :A$ =
    "0123456789ABCDEF": FOR I =  INT ( LOG (A) /  LOG
    (16)) TO 0 STEP  - 1:B =  INT (A / 16 ↑ I): PRINT
     MID$ (A$,B + 1,1);:A = A - (16 ↑ I) * B: NEXT:PRINT
    : GOTO 280
290  REM
300  REM
310  REM BIN)DEC
320  REM
330  INPUT A$:A = 0: FOR I = 1 TO  LEN (A$):A = A +  VAL
    ( MID$ (A$,I,1)) * 2 ↑ ( LEN (A$) - I): NEXT : PRINT
    A: GOTO 330
```

I 6 programmi servono per convertire numeri decimali, esadecimali e binari in qualunque combinazione tra di loro. Di simili ce ne sono tanti, è vero, ma questi hanno la caratteristica di essere stati scritti in modo da occupare una sola riga ciascuno (naturalmente con più di uno statement). Inoltre, permettono la conversione di numeri fino a 32 bit!
Nei calcolatori come l'Apple II, che non ha l'istruzione IF THEN ELSE, non è possibile usare IF THEN per un programma da una riga perché la ELSE (sottointeso) deve necessariamente stare su un'altra riga.
Per convertire una cifra esadecimale in decimale si potrebbe procedere come segue. Supponiamo che A$ contenga la cifra (quindi un carattere tra 0,1,2... 9,A,B,C,D,E,F). Possiamo usare il seguente programma:

```
5  INPUT A$ : REM Input una sola cifra HEX
10 IF VAL(A$)0 THEN A=VAL(A$)
20 IF A$="0" THEN A= 0
30 IF A$="A" THEN A=10
40 IF A$="B" THEN A=11
50 IF A$="C" THEN A=12
60 IF A$="D" THEN A=13
70 IF A$="E" THEN A=14
80 IF A$="F" THEN A=15
90 PRINT A
```

ma non sarebbe possibile condensarlo su una riga. Un metodo più elegante potrebbe essere:

```
10 INPUT A$
20 S$="0123456789ABCDEF"
30 FOR I=1 TO 16
40 IF A$=MID$(S$,I,1) THEN A=I-1
50 NEXT
60 PRINT A
```

ma neanche questo si adatta ad essere scritto su una riga a causa della riga 40 che contiene l'IF. Riscrivendo la riga 40 come segue:

```
40 A = A + (A$=MID$(S$,I,1)) * (I-1)
```

che a prima vista sembra un po' strana, si riesce ad eliminare l'istruzione di IF. Come funziona?... È semplice: quando l'espressione tra parentesi è valida (cioè nel nostro caso quando il carattere di S$ è uguale ad A$), il valore dell'espressione è 1. Se invece l'I-esimo carattere di S$ è diverso da A$, allora l'espressione è falsa e quindi il suo valore è pari a zero. In conclusione A = (I-1) solo quando I è tale che l'I-esimo carattere è uguale ad A$. Avendo eliminato l'istruzione IF possiamo condensare il tutto su un'unica riga come nei programmi pubblicati. Una maxi-riga è pur sempre una riga, no?

Il futuro é dei computer.

8/11 febbraio 1982

EDP USA: L'unica mostra in Italia della piú aggiornata e avanzata produzione americana di computer, peripheral e software compatibile.

EDP USA: Un appuntamento obbligato non solo per gli operatori del settore ma anche per tutti i responsabili di azienda per i quali un'informazione corretta e approfondita nel campo dei computer é ormai d'obbligo.

U.S. INTERNATIONAL MARKETING CENTER

Via Gattamelata 5, 20149 Milano (quartiere Fiera) Telefono (02) 46.96.451, telex 330208 USIMC I.

Ingresso riservato agli operatori del settore, a dirigenti e professionisti. Orario continuato dalle 9 alle 18.

CAD/CAM (Computer-Aided Design /Computer-Aided Manufacturing)

In collaborazione con la rivista **PIXEL** a latere della mostra si terrà una serie di conferenze tecniche sul tema specifico, tenute dagli esperti più qualificati del settore, italiani e stranieri.