Ricerche in tabella

di Anna Pugliese

Il problema delle ricerche in tabella nasce tutte le volte che ci si trova davanti ad un insieme di dati interessanti soprattutto dal punto di vista dell'informazione ed alla necessità di reperire, in mezzo ad essi, quello cercato. Si pensi al proposito, ad uno schedario o ad un dizionario, in generale ad un elenco elettronico comprendente una, più o meno grossa, mole di dati

È noto come ogni elemento di una tabella sia composto da una parte detta «*chiave*» e da un'altra detta «*informazione*» associata alla chiave. Un banale esempio di tabella è riportato in figura 1.

Esempi di campi di applicazione di simili problematiche, possono essere:
— problemi scientifici (tavole matematiche, classificazioni);
— gestione anagrafica;
— applicazioni bancarie e commerciali in genere;
— archiviazioni;
— programmazione di base.

A proposito di quest'ultimo punto, si pensi ai compilatori ed alla necessità, che essi hanno, di mantenere e reperire il più velocemente possibile, elementi appartenenti alla tavola dei simboli (un elenco di tutti i nomi usati dal programma e degli oggetti che tali nomi rappresentano).

I vari campi applicativi presentano certamente caratteristiche diverse, e lo stesso si può dire per problemi dello stesso tipo ma che diversificano da caso a caso. Emerge tuttavia una comune impostazione del problema, al quale possono essere applicate soluzioni diverse, ognuna delle quali fissa un valore al rapporto *costo/prestazioni* che diventa a questo punto il vero nocciolo del problema. Più concretamente si possono considerare come elementi diversificanti, quelle elencati in figura 2.

CHIAVE	INFORMAZIONE
AGRIGENTO	92100
ALESSANDRIA	15100
ANCONA	60100
AOSTA	11100
AREZZO	52100
A.PICENO	63100
ASTI	14100
AVELLINO	83100
BARI	70100

Figura 1 - Un esempio di tabella.

Per trattare il problema in maniera organica, e giungere così alla presentazione delle varie tecniche di risoluzione dello stesso, è opportuno spendere due parole nella definizione di una terminologia appropriata. Innanzitutto, con il termine *Tabella* ci si riferirà ad una qualsiasi successione di elementi del tipo:

$E_i=(K_i, I_i)$

dove K_i è la chiave dell'elemento E_i ed I_i è l'informazione ad esso associata. Per chiarire la cosa, si può pensare ad un elemento in cui la chiave corrisponde al nome di una variabile, e l'informazione corrisponde all'indirizzo di memoria associato alla variabile stessa.

Parlando di successione di elementi, ciò che si intende non è legato al modo in cui tali elementi sono effettivamente memorizzati; in altri termini: l'allocazione in memoria degli elementi della tabella può essere ottenuta sia mediante l'impiego di strutture a lista, che mediante un semplice array; in realtà la scelta fra queste due tecniche di memorizzazione non è completamente libera, ma legata al tipo di soluzione adottata.

Riguardo al supporto di memorizzazione utilizzato per la tabella, abbiamo già accennato (figura 2) all'influenza che la scelta ha sul metodo effettivo di ricerca utilizzabile. D'ora in avanti sarà considerato il solo caso della memoria principale, essendo l'utilizzo della memoria secondaria pesantemente limitativo nei confronti di una ricerca efficiente. Apparirà subito chiaro, come tale requisito non sia affatto necessario nel caso del primo metodo di ricerca che sarà illustrato.

Una volta stabilita l'esistenza di una tabella, sotto forma di successione di elementi, il passo successivo riguarda il tipo di operazioni che, sulla tabella, possono essere effettuate. L'operazione più importante è chiaramente quella di ricerca dell'informazione associata ad

1 Grandezza della tabella (tabelle lunghe/corte)
2 Supporto di memorizzazione usato (memoria principale di massa)
3 Prevedibilità del dimensionamento (tabelle fisse/variabili)
4 Prevedibilità della distribuzione delle chiavi (chiavi conosciute a-priori/sconosciute).

Figura 2 - Elementi che contribuiscono alla scelta di un particolare metodo di ricerca in tabella.

un certo elemento, nota la chiave che lo contraddistingue. Tale operazione di ricerca, potrebbe poi essere seguita, ad esempio, da una rimozione dell'elemento della tabella, oppure da una inserzione dell'elemento stesso, qualora esso fosse risultato assente. Essendo la ricerca l'operazione di maggiore interesse, viene spontaneo adottare come misura della bontà del metodo di ricerca adottato, una quantità detta *lunghezza media di ricerca*, ed indicata con il simbolo «S» (to search=cercare), indicante il numero medio di passi necessari ad individuare un elemento in tabella. Tale lunghezza media di ricerca è calcolata a partire dalle varie lunghezze di ricerca dei singoli elementi: detto S_i il numero di passi necessario per individuare l'i-esimo elemento in tabella, si ha:

$$S=(S_1/n+S_2/n+...+S_n/n)$$

dove n è il numero di elementi della tabella.

Ricerca completa

La prima tecnica di ricerca in tabella che viene in mente, è certamente quella che consiste nello scorrere sequenzialmente tutti gli elementi della tabella fino all'individuazione di quello la cui chiave corrisponde a quella cercata. Data una tabella T di n elementi, basterà allora utilizzare un array di n elementi; nel caso di tabella variabile sarà necessario effettuare una stima del massimo numero di elementi previsto, ed effettuare in base ad esso il dimensionamento dell'array. Una soluzione alternativa potrebbe essere quella di memorizzare a lista la tabella, ed utilizzare allocazione dinamica per i nuovi elementi della lista; in quest'ultimo caso anche la rimozione sarebbe facilmente effettuabile, mentre nel caso di un array non resta che effettuare solo la rimozione logica dell'elemento, che corrisponde a marcare con un valore speciale l'elemento da rimuovere. Per quanto riguarda la lunghezza di ricerca, si ha che:

$S_i=i$ per $i=1,2,...,n$

ed

$$S=(1/n)*(S_1+S_2+...+S_n)=(n+1)/2$$

che è la più lunga possibile. Essa può essere ridotta nel caso in cui siano noti a-priori gli elementi più frequentemente ricercati, ponendoli nelle prime posizioni della tabella. Resta invece irriducibile il numero di passi necessario per concludere l'inesistenza di un elemento in tabella, che è pari ad n.

Ricerca binaria (o logaritmica)

Un sensibile miglioramento della lunghezza media di ricerca può essere ottenuto utilizzando un metodo che tenga conto di un ordinamento fra le chiavi degli elementi in tabella. Purché si faccia uso di un ordinamento totale, poco importa su quali basi tale ordinamento è stabilito; ciò nonostante è d'obbligo considerare quello che è il più diffuso fra gli ordinamenti: l'alfanumerico.

Il metodo di ricerca cui ci si riferisce è basato dunque sull'assunzione che gli elementi siano stati inseriti nella tabella, rispettando l'ordinamento delle loro chiavi; così, per restare nel caso dell'esempio della figura 1, l'elemento con chiave «Roma» si troverà nella tabella prima di quello con chiave «Torino» ma dopo quello con chiave «Milano», vale a dire che apparterrà alla sottotabella di elementi compresi tra «Milano» e «Torino».

Il metodo in questione, che prende il nome di «Ricerca Binaria», consiste essenzialmente nell'accedere come primo passo, all'elemento centrale della tabella, confrontare la chiave di tale elemento con quella ricercata, e proseguire la ricerca, utilizzando la stessa strategia, nella sottotabella precedente, o in quella seguente, l'elemento individuato, in dipendenza dell'esito del confronto effettuato, fino al ritrovamento dell'elemento cercato. Il nome di questa strategia di ricerca deriva da questo continuo dividere in due parti la tabella; l'alternativo nome di «Ricerca Logaritmica» deriva, come si vedrà, dal valore della lunghezza media di ricerca che risulta essere dell'ordine del logaritmo della grandezza della tabella.

Al fine di osservare in maggior dettaglio la funzionalità del metodo, ci si ponga anzitutto nell'ipotesi di avere utilizzato come struttura informativa per la tabella, un array di n elementi. Si consideri l'esempio riportato in figura 3; esso mostra un array bidimensionale V, di 13 righe e 2 colonne, dove ogni riga contiene nella prima colonna la chiave di un

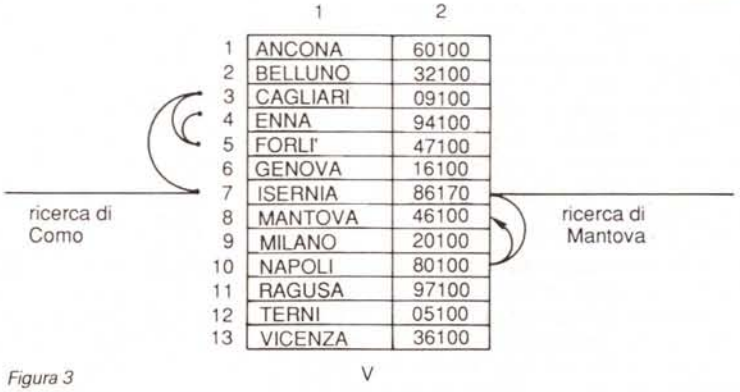

Figura 3

```
Function Binary_Search (a,b, : integer ; k : string) : integer
begin
var m:integer
        m:=INT( (a+b)/2 ) ;
        if V[m,1]=k then return m ;
        if a≥b      then return -1 ;
        if V[m,1]<k  then return (Binary_Search(m+1,b,k))
                     else return (Binary_Search(a,m-1,k)) ;
end ;
```

Figura 4

certo elemento, e nella seconda l'informazione ad essa associata. Sia il C.A.P. di Mantova l'informazione oggetto della ricerca. Gli archi riportati sulla destra della tabella in figura 3, descrivono il percorso di ricerca da effettuare per ritrovare la chiave «Mantova» nella tabella, mentre l'algoritmo di ricerca è mostrato in figura 4.

Si consideri adesso la ricerca della chiave «Como». Al primo passo si confronterà «Como» con V[7,1], al secondo con V[3,1], elemento di mezzo della sottotabella che precede V[7,1], al terzo con V[5,1] ed al quarto con V[4,1]. Gli archi riportati sulla sinistra della tabella in figura 3 riassumono questa successione. L'inesistenza della chiave «Como» in tabella, è risolta al quarto passo quando Binary_Search è chiamata con a=b=4 e si ha V[4,1]<>«Como».

La ricerca binaria, così come è stata presentata, è applicabile solo nel caso di tabelle fisse. Per le tabelle variabili (quelle cioè sulle quali sono possibili inserzioni e rimozioni di elementi) è necessario ricorrere ad una memorizzazione della tabella mediante un albero binario. Per maggiori dettagli al riguardo, si rimanda il lettore interessato agli ultimi numeri della rubrica, dove sono stati presentati esempi di ricerca sugli alberi binari.

Per ciò che concerne la lunghezza media di ricerca si osservi che nell'esempio fatto, in cui n=13, si è ottenuto:

$S_{Mantova}$= 3 ed S'=4

dove con S' si è indicata la lunghezza di ricerca di una chiave inesistente (nell'esempio: «Como»). È possibile dimostrare che la lunghezza media della ricerca binaria in tabelle di n elementi è dell'ordine di $\log_2(n)$. Per n=13 si ha dunque: $S=\log_2(13)$ che è un numero compreso tra 3 e 4.

Codifica hash

Con il nome di *TABELLE HASH* (o Randomized Tables) ci si riferisce nella letteratura a tabelle sulle quali viene applicata una particolare strategia di ricerca, basata sull'esistenza di una decodifica della chiave cercata, che permette di ottenere la massima quantità di informazione possibile sulla posizione dell'elemento associato ad essa. Sia K_i la chiave cercata, e siano m i bit da cui è costituita la sua codifica binaria. Ora, di chiavi composte esattamente da m bit, si sa, ne esistono esattamente 2^m. È chiaro allora che utilizzando un vettore lungo $n=2^m$, come struttura per la tabella, è possibile inserire ciascun elemento in una ben precisa posizione, quella ad esempio, corrispondente al valore della codifica binaria della loro chiave. In tal modo, detta h la funzione che trasforma una chiave nella sua codifica binaria, l'elemento cercato, di chiave K_i, si troverà in $V[h(K_i)]$.

La lunghezza media di ricerca di questo metodo è S=1 che è la più bassa possibile. Le tabelle organizzate in modo tale che applicando un'opportuna funzione sulla chiave, se ne determina univocamente la posizione, vengono dette *tabelle ad accesso diretto*. Con tali tabelle, non è richiesto il confronto tra la chiave cercata e quella degli altri elementi, cosicché non è necessario memorizzare le chiavi assieme ai corrispondenti elementi nella tabella.

Le tabelle ad accesso diretto costituiscono un caso particolare delle tabelle hash; per queste ultime non è richiesto infatti, che la funzione hash determini univocamente la posizione dell'elemento associato, permettendo in tal modo di utilizzare chiavi più generiche.

In tabelle hash di tipo generale è possibile dunque che si verifichino delle «collisioni» fra elementi, vale a dire che date due chiavi K_i e K_j è possibile che sia $h(K_i)=h(K_j)$. Per capire come tali collisioni possono essere trattate, conviene considerare quello che succede usando la tecnica della funzione hash nell'operazione di inserimento degli elementi in tabella. Si consideri allora un certo insieme K di chiavi che vanno inserite in una tabella hash. Dato K, e data la funzione h, è possibile partizionare K in s sottoinsiemi, ponendo nello stesso sottoinsieme tutte le chiavi sulle quali l'applicazione di h produce uno stesso valore. Alcuni di tali sottoinsiemi conterranno esattamente 1 chiave, altri più chiavi, qualcuno nessuna. Sia K_i una chiave da inserire. Calcolata $h(K_i)$ occorrerà controllare, prima che avvenga l'inserimento, se in $V[h(K_i)]$ è già presente un altro elemento. Se questo è il caso, è necessario provvedere alla ricerca di una posizione libera nel vettore. La ricerca di tale posizione viene effettuata a partire dall'indirizzo $h(K_i)$, sommando ad esso un particolare valore detto «incremento». Si osservi la figura 5.

ANCONA	1	ANCONA	60100
TORINO	2	VERONA	37100
GENOVA	3	CAGLIARI	09100
CAGLIARI	4	BRESCIA	25100
VICENZA	5		
VARESE	6		
VERONA	7	GENOVA	16100
BRESCIA	8		
	9		
	10		
	11		
	12		
	13		
	14		
	15		
	16		
	17		
	18	TORINO	10100
	19		
	20	VICENZA	36100
	21	VARESE	21100

Figura 5

Essa mostra un vettore di 21 elementi sul quale viene memorizzata una tabella di alcuni C.A.P. di città italiane. La funzione hash scelta, associa ad ogni chiave il numero che esprime la posizione dell'iniziale della chiave nell'alfabeto italiano. L'ordine con cui i vari elementi sono stati inseriti nella tabella è quello mostrato nella lista di città riportata sempre in figura 5. Il primo elemento da inserire, «Ancona», va nel primo elemento del vettore, essendo h(«Ancona»)=1 ed essendo libera la posizione 1 del vettore. Le inserzioni procedono senza problemi fino alla chiave «Vicenza». Giunti a «Varese» abbiamo la prima «collisione», precisamente tra «Varese» e «Vicenza». Il modo più semplice di risolvere il problema è quello di cercare la prima posizione successiva che sia ancora libera; quindi la 21. Nuova collisione nel caso di «Verona», che collide prima con «Varese» poi con «Vicenza» ed infine con la legittima «Ancona», che occupa la prima posizione libera successiva a «Vicenza»; «Verona» finisce dunque nella posizione 2, rubando il posto a «Brescia» che essendo arrivata tardi dovrà accomodarsi nella posizione 4.

L'esempio fatto dovrebbe chiarire un po' il funzionamento delle tabelle hash. L'argomento tuttavia non si esaurisce qui. Dedicheremo il prossimo numero della rubrica alla trattazione delle varie funzioni hash esistenti e delle possibili «leggi di scansione» che possono esservi applicate.

PRODOTTI PROFESSIONALI PER LA GRAFICA

CAD EXPRESS

CAD WORKSTATIONS

GS-300: CPU 386/20, 1 Mb RAM, 40 Mb H.D., VGA, Monitor 14'' Multisync L. **7.500**

GS-2000: CPU 286/12, 1 Mb RAM, 20 Mb H.D., Scheda Leonard 1024 × 768 n.i., Monitor ADI colore 100 MHz 19'', Monitor operativo b/n 12'' L. **9.900**

GS-3000: CPU 386/20, 1 Mb RAM, 40 Mb H.D., Scheda Leonard 1024 × 768 n.i., Monitor ADI colore 100 MHz 19'', Monitor operativo b/n 12'' L. **12.000**

GS-3000/25: CPU 386/25, 2 Mb RAM, 64 Kb Cache, 150 Mb H. D., Scheda Leonard 1024 × 768 n.i., Monitor ADI colore 100 MHz 19'', Monitor operativo b/n 12'' L. **17.000**

ACCEL-500

LA PIÙ VERSATILE STAMPANTE OGGI SUL MERCATO.

Emulazione plotter fino al formato A2 in HP-GL, a colori.

Stampante 24 aghi, 480 cps, a colori, 6 emulazioni, 4 fonts residenti con possibilità di aggiunta schedine per fonts ed emulazioni opzionali, incluse Tektronix, DEC, MacIntosh L. **3.400**

SOFTWARE E CAD 3-D

- CAD 3-D altamente specializzato, per architetti.
- Computo Metrico, Topografia, Strade, Strutture, Cantieri.
- Sofware per DTP, Arts & Letters, Clip Arts.

VISION 16

Scheda per acquisizione di immagini da telecamera, 32768 colori, fornita con sofware Colorscheme 1 L. 2.500

Optionals: Telecamera JVC RGB/PAL, Videoprinter Hitachi, Convertitore RGB e viceversa - segnale composito. Software: Colorscheme 2, Freestyle, Halo 88, PC Album, Lumena, Crystal 3-D, TEGA/T-SCAN...

GRAPHICS CARDS PER PC E PS/2

Leo VGA: da 320 × 200 a 800 × 600, 256 colori L. **980**

Leo VGA + : 800 × 600, 16 colori L. **530**

Leonard 1 + : 1024 × 768, compat. Artist 1 Plus L. **2.200**

Metheus UGA 1104: 1024 × 768, emul. CGA, 50 milioni pix/sec. L. **3.200**

Metheus UGA 1124: 1024 × 768, emul. CGA, EGA, VGA, 50 milioni pix/sec. L. **4.200**

Metheus UGA 1128: 256 colori, emul. EGA, VGA, 50 milioni pix/sec. L. **5.750**

SPEA BOARDS per AT BUS:

Painter P1: 1024 × 768 n.i., 16 colori, 512K RAM L. **2.600**

Painter P3: 1024 × 768 n.i., 256 colori, 1 Mb RAM L. **3.990**

Gallery S: 1280 × 1024, 16 colori, 1 Mb RAM L. **4.300**

Gallery 2: 1280 × 1024 n.i., 256 col., 2 Mb RAM L. **7.000**

SPEA BOARDS per PS/2:

Flash 1P: 1024 × 768 n.i., 256 colori, 1 Mb RAM L. **3.990**

Flash 1GS: 1280 × 1024 n.i., 16 colori, 1 Mb RAM L. **4.700**

Flash 1G2: 1280 × 1024 n.i., 256 colori, 2 Mb RAM L. **7.250**

Flash 2: 1280 × 1024 n.i., 16 colori, 1 Mb RAM L. **6.150**

Flash 3-12: 1280 × 1024 n.i., 16 colori, 2 Mb RAM L. **8.000**

Flash 3/16: 1600 × 1280 n.i., 16 colori, 2 Mb RAM L. **8.400**

PLOTTERS

Ioline LP-3500: A1, 1 penna, 25 cm/sec L. **7.200**

Ioline LP-3700: 200 × 90 cm., 8 penne, 25 cm/sec L. **9.500**

Ioline LP-4000: 200 × 90 cm., 8 penne, 50 cm/sec L. **11.500**

Enter SP-600: A3/A4, 6 penne, 35 cm/sec L. **2.000**

Enter SP-1800: A1, 8 penne, 80 cm/sec L. **8.500**

Enter SP-2800: A0, 8 penne, 60 cm/sec L. **12.000**

Mutoh IP500EL: A0, 8 penne/8 mine, 50 cm/sec L. **12.500**

Mutoh F-910E: A0, 8 penne/40 mine, 113 cm/sec L. **17.000**

PLOTTERS SPECIALI

- Per taglio vinile, films, carta, cartone, materiali plastici ecc.
- Per cartellonistica, pubblicità: con punte speciali a caldo, laser o diamante, pennarelli, penne per camera oscura.
- Per sartoria, cartografia e applicazioni industriali: formato carta mt. 1,80 × 300.

CHIEDETE INFORMAZIONI SPECIFICHE!

PLOTSERVER PLUS

RISPARMIATE IL VOSTRO TEMPO!

Il PLOTSERVER è un dispositivo interfacciabile con qualunque plotter o stampante, che accetta Floppy Disks da 3,5'' o 5,25'' e permette di far lavorare autonomamente la periferica, liberando il computer dai tempi di attesa.

Chiedete informazioni!

MONITORS

ADI 14'' b/n	L.	**150**
ADI 14'' colore, Multisync	L.	**1.500**
ADI 19'' colore, 100 MHz, 50 KHz	L	**4.250**
SONY 20'' Trinitron colore, 1024 × 768 o 1280 × 1024	L.	**9.800**

CONDIZIONI

- Prezzi in migliaia.
- I.V.A. esclusa.
- Franco ns. sede.
- CONSEGNA IMMEDIATA.

EXPRESS CAD

Via Bellaria, 54 - Pistoia - Tel. 0573/368113