Ricerche in tabella (2)

Funzioni e Incrementi hash

di Anna Pugliese

Il numero di targa della mia automobile, comincia con un 4 e finisce col suo doppio, e come cifre intermedie ha tutti i numeri primi, più piccoli di 10, messi in ordine crescente. Per ricordare la data di nascita di un mio amico (acc... ora sapete anche la mia data di nascita, ndadp), prendo una calcolatrice e calcolo la radice di due: in mezzo alle sue prime cifre decimali è presente la seguenza cercata (13/5/62).
Ora, io so quanto matematici siano gli stratagemmi che usate voi per ricordare simili cose, ma, certamente, un po' tutti ricorriamo non di rado a simili trucchetti. Cosa diavolo c'entra tutto questo? È presto detto: stratagemmi o trucchi che dir si voglia, un meccanismo che trasformi un'informazione «orecchiabile» in un freddo ed austero dato, non è altro che una funzione hash

Nel campo dell'archiviazione di elementi, una funzione hash ha lo scopo di produrre come risultato, quando viene applicata ad una «chiave», l'indirizzo presunto dell'elemento associato alla chiave, relativamente ad una struttura informativa ad accesso diretto, nella quale gli elementi dell'archivio sono stati memorizzati.

Un archivio elettronico di dati, si sa, è un oggetto sul quale possono essere effettuate operazioni di tre tipi:
1) Inserzione di nuovi dati
2) Rimozione di dati obsoleti
3) Consultazione del valore di un dato.

Tutte e tre queste operazioni potrebbero essere scomposte in modo da evidenziare l'esistenza di una sottoperazione comune, vale a dire: l'individuazione del valore di un dato all'interno dell'archivio.

A motivo, tanto di questa sua fondamentalità, quanto della sua influenza, crescente al crescere delle dimensioni dell'archivio, sulla durata complessiva dell'operazione, la sottoperazione considerata è stata ed è a tutt'oggi, oggetto di studi approfonditi. Nel contesto delineato essa ha preso il nome di «ricerca in tabella».

L'efficienza di un'operazione di ricerca in tabella, requisito indispensabile nei grossi archivi di dati, può essere conseguita mixando opportunamente una buona organizzazione della struttura informativa che implementa l'archivio, ed un'efficiente strategia di ricerca. Molto spesso, per efficiente strategia di ricerca, può intendersi: una buona funzione hash.

Il problema della collisione degli elementi

Nella prima parte di questa trattazione, è stato riportato un esempio per illustrare la funzionalità di una «Randomized Table» (una tabella cui è applicata la ricerca mediante funzione hash). In questo paragrafo è il caso di riportare alcuni dei concetti presentati allora, inquadrandoli nell'ottica della costruzione di una vera funzione hash.

Il «tallone d'Achille» delle funzioni hash, è rappresentato da un fenomeno che prende il nome di «collisione di elementi». Due elementi di una tabella hash, si dicono collidere, quando coincide il valore prodotto come risultato dall'applicazione della funzione hash sulle loro chiavi (la chiave di un elemento è un identificatore, unico, dell'elemento, a partire dal quale la ricerca dell'elemento stesso è condotta).

Data l'univocità delle chiavi, è teoricamente possibile utilizzare una funzione hash che non provoca collisioni (basta scegliere una funzione iniettiva). In tal caso la funzione hash è caratterizzata dal rispetto dei due seguenti requisiti:

$$1)\ \text{hash}(k_i) = j \qquad 1 \leq j \leq l$$

$$2)\ K_x \neq K_y \rightarrow \text{hash}(K_x) \neq \text{ash}(K_y)$$

dove con l si è indicata la lunghezza complessiva della tabella. Le funzioni che rispettano i requisiti 1 e 2, sono dette funzioni hash ad «accesso diretto». L'inconveniente di cui soffre tale soluzione è dovuto alla necessità di dimensionare la tabella, in modo che esista una posizione, in essa, per ogni possibile combinazione dei caratteri delle chiavi.

Nel caso in cui le chiavi degli elementi siano costituite da un codice numerico, è spesso possibile trovare una soluzione opportuna. Si supponga ad esempio, che l'archivio debba mantenere all'incirca 80.000 registrazioni; basterà allora utilizzare chiavi formate da 5 cifre decimali, ed utilizzare le stesse direttamente come indirizzo, nella tabella, della registrazione corrispondente alla chiave. È superfluo sottolineare gli inconvenienti pratici di cui soffre l'accesso ad un archivio, per codice di registrazione. L'utente di un sistema elettronico di

archiviazione, è spesso stanco persino di digitare l'intero nome e cognome della persona di cui vuole ritrovare alcuni dati. Si pensi allora, tanto per non sbilanciarsi troppo, ad una chiave d'accesso per Codice Fiscale; questi ultimi sono costituiti da una sequenza di 16 caratteri alfanumerici di cui l'ultimo è ridondante. I possibili codici fiscali, il cui numero esatto è difficile da calcolare, si aggirano all'incirca su un numero che è dell'ordine della quindicesima potenza di 26.

Senza bisogno di far impazzire la calcolatrice, si noti che l'ottava potenza di 26 è approssimativamente 2500 miliardi. Vale a dire che, anche ad essere infinitamente generosi, se il Ministero delle Finanze usasse per archivio, una tabella ad accesso diretto con il Codice Fiscale per chiave, dovrebbe dimensionare l'archivio stesso a 3500 volte l'intero numero degli abitanti della Nazione.

Con questo non si creda che le tabelle ad accesso diretto siano praticamente inapplicabili. Tutt'altro. Esse, tuttavia, richiedono delle assunzioni da farsi sulla composizione delle chiavi, che esulano dal contesto presente.

Le tabelle hash, di cui quelle ad accesso diretto sono solo un caso particolare, non rispettano il requisito di iniettività, e possono dunque generare delle collisioni. Per limitare i danni prodotti da tali collisioni, è necessario utilizzare funzioni che diano una buona distribuzione degli indirizzi prodotti. Una volta rilevata, durante una ricerca in tabella, una collisione di elementi, la ricerca deve proseguire in altre posizioni della tabella, fino al ritrovamento dell'effettivo elemento cercato. La strategia mediante la quale è condotta questa ulteriore ricerca, è detta «legge di incremento», e deve essere tale da produrre una sequenza di indirizzi alternativi per l'elemento cercato, senza che tale sequenza rigeneri indirizzi già presi in considerazione.

Generazione di un indirizzo hash

La parola «hash», sta ad indicare un impasto fatto da molti ingredienti tritati assieme. Per gli americani, questo nome, nel caso delle funzioni hash, è significativo di per se stesso, e rappresenta qualcosa di simile ad una «polpetta di bit».

I due più semplici metodi per ottenere questa polpetta, data una chiave K_i, considerata come una semplice sequenza di bit (quindi già codificata in codice binario a partire dalla sua presumibile natura alfanumerica), sono i seguenti:

1) hash (K_i) è costituito dagli n bit centrali della chiave K_i;

2) hash (K_i) è costituito dalla somma di più sottoinsiemi, di n bit, della chiave K_i.

Un terzo metodo, più raffinato dei precedenti, è il seguente: 3) detta la lunghezza della tabella, hash (K_i) è dato dal resto della divisione tra K_i ed l.

Adottando i metodi 1 e 2, è opportuno definire la lunghezza l della tabella come una potenza di 2, e scegliere n come la sua potenza; vale a dire:

$$l = 2^n.$$

La sequenza di n bit ottenuta come risultato della funzione hash, sarà dunque un numero compreso tra 0 ed l-1, ed individuerà la posizione, all'interno del vettore che implementa la tabella, dell'elemento associato alla chiave di ricerca.

Il caso della funzione ottenuta con il terzo metodo, deve invece essere applicato con una tabella di lunghezza L che non sia una potenza di 2, per evitare che l'indirizzo prodotto presenti gli stessi bit finali della chiave; scegliendo L come un numero dispari si ottengono risultati ancora migliori. Una volta effettuata la divisione fra la chiave L, si avrà:

$$\frac{K_i}{L} = Q + \frac{R}{L}$$

e si sceglierà hash (K_i) = R;
l'indirizzo ottenuto sarà ovviamente compreso tra 0 ed L-1, ed anche in questo caso individuerà la posizione, all'interno del vettore che implementa la tabella, dell'elemento di chiave K_i.

Nonostante il metodo 3 richieda l'esecuzione della divisione tra K_i ed L (operazione molto costosa, in termini di tempo, rispetto ad una semplice somma), l'uniformità di distribuzione degli indirizzi prodotti è molto elevata, assicurando così la riduzione di agglomerati di elementi e quindi, indirettamente, delle collisioni fra gli stessi.

Per esemplificare il procedimento di ricerca in una tabella hash, si consideri la ricerca dell'elemento associato alla chiave K_i.
hash (K_i) produce l'indirizzo di una posizione della tabella che «dovrebbe» contenere l'elemento cercato. Tuttavia, è possibile che tale posizione sia già stata occupata da un elemento, di chiave K_j, per il quale si è, ad esempio, ottenuto:

$$\text{hash } (K_j) = \text{hash } (K_i)$$

Una volta generato hash (K_i), si dovrà allora confrontare K_i con la chiave dell'elemento presente effettivamente in tabella. Se tale chiave è diversa da K_i è segno che si è ottenuta una collisione, e la ricerca deve proseguire in qualche modo, con la generazione di un nuovo indirizzo.

Subentra a questo punto la cosiddetta «legge di incremento».

Tale legge permette di valutare, ad ogni passo, un incremento p_k rispetto all'indirizzo prodotto dalla funzione hash, dove continuare la ricerca dell'elemento associato alla chiave data.

In altri termini, il meccanismo di ricerca in una tabella hash, consiste nel cercare la chiave in una serie di posizioni della tabella, dove la prima posizione è data da:

$$h_0(K_i) = \text{hash } (K_i)$$

mentre le posizioni successive sono date da:

$$h_1(K_i) = [\text{hash}(K_i) + p_1] \text{ mod. } l$$
$$h_2(K_i) = [\text{hash}(K_i) + p_2] \text{ mod. } l$$
. . . . . .

L'operazione «modulo l», permette di utilizzare il vettore che implementa la tabella, come una struttura circolare, e garantisce che l'indirizzo prodotto sarà sempre un numero compreso tra 0 ed l−1.

È evidente che una volta tentato l'accesso all'elemento posto all'indirizzo

$$h_{l-1}(K_i) = [\text{hash } (K_i) + p_{l-1}] \text{ mod. } l$$

che sarà certamente l'ultimo elemento non ancora ispezionato nella tabella, si potranno concludere due fatti: il primo, ovvio, è che la chiave K_i non è presente in tabella; il secondo è che la tabella è completamente piena di elementi.

Questo secondo fatto non dovrebbe mai accadere, perché, come vedremo, una tabella piena degrada moltissimo l'efficienza del meccanismo di ricerca. In tabelle non completamente riempite di elementi, l'assenza di un elemento dalla tabella può essere stabilito non appena si ottiene, dalla sequenza prodotta, una posizione del vettore che non contiene alcuna chiave, quindi, nella migliore delle ipotesi, potrebbe bastare addirittura il primo accesso in tabella, per concluderne l'inesistenza in essa dell'elemento cercato.

Incrementi hash

Il più semplice metodo per scorrere, a partire dall'indirizzo ottenuto dalla funzione hash, l'intero vettore, consiste nell'utilizzare un incremento fisso unitario. Il K-esimo indirizzo prodotto dalla sequenza hash utilizzando un incremento unitario, può essere calcolato mediante la seguente espressione:

$$h_k(K_i) = [\text{hash}(K_i + k] \text{ mod. } l$$
$$\text{con } k = 1, 2, 3, \ldots\ldots, l-1.$$

Nella sua forma generale questo metodo è detto *Incremento Lineare* ed è calcolato dalla seguente espressione:

$$h_k(K_i) = [\text{hash}(K_i + q*k] \text{ mod. } l$$

incrementando in tal modo, di q posizioni alla volta. Per garantire l'esplorazione di tutte le posizioni del vettore, occorre adottare un valore per q, che sia primo con il valore di l.

Il complessivo metodo di ricerca che ne viene fuori, presenta un'efficienza

che è valutabile attraverso la misura della lunghezza media di ricerca (numero medio di accessi in tabella da fare per poter reperire l'informazione associata alla chiave cercata). Dal calcolo di tale lunghezza media di ricerca, indicata con S, emerge che essa è funzione della percentuale di occupazione della tabella. Detto T il numero totale di elementi presenti nella tabella, lunga L posizioni, la percentuale di occupazione della tabella è data dal rapporto T/L. Indicando con P tale rapporto, si ha che:

$$S(P) = \frac{1-(P/2)}{1-P}$$

Questo fatto significa che all'aumentare delle dimensioni di un archivio di dati, non necessariamente il tempo di ricerca aumenta con esse; scegliendo infatti di pagare con una moneta che si chiama «memoria occupata» (o, «memoria sprecata»), è possibile mantenere le prestazioni del sistema ai livelli desiderati. Utilizzando una tabella occupata solo per 3/4 della sua effettiva capacità, la lunghezza media di ricerca sarà: S(0,75)=2,5 vale a dire che per accedere ad un generico elemento della tabella, basteranno sempre due o tre tentativi. E si può ottenere anche di meglio, applicando leggi di incremento più sofisticate.

L'impatto della legge di incremento adottata, sulla lunghezza media di ricerca del metodo che ne deriva, può essere compresa riflettendo sul seguente esempio.

Siano K_a, K_b, e K_c tre chiavi collidenti; vale a dire:

$$hash(K_a) = hash(K_b) = hash\ (K_c).$$

Avendo inserito le tre chiavi in tabella, si sarà ottenuto, ad esempio:

K_a inserita all'indirizzo $h_0(K_a) = hash(K_a)$,
K_b inserita all'indirizzo $h_1(K_b) = hash(K_b)+p_1$,
K_c inserita all'indirizzo $h_2(K_c) = hash(K_c)+p_2$.

Sia K_d una chiave per cui: $hash(K_d) = hash(K_b)+p_1$.

K_b e K_d non collidono direttamente, ma indirettamente sì.

In altri termini, K_b occupa la legittima posizione di K_d. Dovendo inserire quest'ultima in tabella, si tenterà di inserirla dapprima in

$$h_0(K_d) = hash(K_d) = hash(K_b)+p_1.$$

Il tentativo sarà vano perché la posizione risulterà già occupata. Al secondo tentativo si proverà all'indirizzo $h_1(K_d) = hash(K_d) + p_1$. Se è stato adottato un incremento fisso unitario, tale posizione sarà già occupata da K_c, infatti:

$$h_1(K_d) = hash(K_d)+p_1 = (hash(K_b)+p_1 + p_1 = = hash(K_b)+p_2 = hash(K_c)+p_2 = h_2(K_c).$$

Sarà dunque necessario un terzo tentativo. Questi 3 accessi si ripeteranno ad ogni ricerca dell'informazione associata a K_d e rappresentano, quindi, un handicap che non sarebbe esistito con una legge di incremento variabile invece che fisso e/o una legge di incremento che è funzione dell'indirizzo ottenuto applicando la funzione hash sulla chiave.

I metodi che derivano da queste considerazioni sono svariati, e vanno dalla legge di incremento «quadratico» a quella «pseudocasuale» a quella «pesata». Per comprendere la bontà di queste sofisticate leggi di incremento può bastare riportarne il valore di S(0,75) che è pari circa ad 1,8 contro il 2,5 dell'incremento lineare.

Reperire velocemente i dati, si può!