Corso di Informatica

Modulo T1

E1–Organizzazione hash

Prerequisiti

È necessario conoscere:

- concetto matematico di funzione
- distribuzione uniforme

Informazioni generali

In questa Unità si introduce un altro tipo di *organizzazione non sequenziale* degli archivi, detta **organizzazione hash**.

Si tratta di un tipo di organizzazione utilizzata di frequente e che consente un'alta velocità nelle operazioni di ricerca e di aggiornamento dei dati:

- in memoria di massa (archivi);
- in memoria centrale (tabelle)

Ne analizziamo, come sempre, vantaggi e svantaggi.

Organizzazione *hash*

Gli archivi nei quali è possibile l'accesso diretto, possono essere progettati secondo l'**organizzazione hash**.

Questa tecnica, detta anche **hashing**, consiste nell'associare a ciascun record una chiave, che viene successivamente trasformata in un numero, tramite un'apposita funzione, detta appunto **funzione hash**.

Questo numero viene usato per indicizzare i record del file ed è pertanto il **R.R.N**.

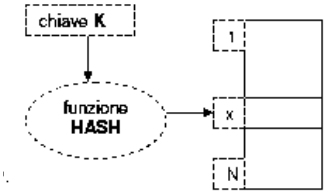
La funzione *hash*

La funzione che cerchiamo è una funzione del tipo:

$$\mathbf{f}: S_K \rightarrow S_I$$

dove

- S_K è l'insieme detto **spazio delle chiavi**
- S_I è lo **spazio degli indirizzi**, ad esempio {1, ..., N}



Trasformazione perfetta

Una funzione *hash* dovrebbe averre le seguenti proprietà (1 ed N indicano gli estremi dell'intervallo degli R.R.N.):

1) $\mathbf{f}(K_i) = R$ (con $R \in S_i$, $K_i \in S_K$)

2) $\mathbf{f}(K_i) \neq \mathbf{f}(K_j)$ (per ogni $K_i \neq K_j$, K_i, $K_j \in S_K$) (iniettiva)

3) **f**() è completa (copre da 1 a N) (suriettiva)

Una funzione avente i requisiti 1) …3) risulta biunivoca e si dice **trasformazione perfetta**. In questo caso ogni dato presente in tabella può essere reperito mediante un unico accesso; è un esempio di accesso casuale.

Trasformazione perfetta

La trasformazione perfetta è un caso ideale, che in pratica non conviene realizzare mai.

ATTIVITA': ad esempio, nel caso banale K ϵ [A,B,C,…,Z], considerando chiavi di soli 5 caratteri alfabetici inglesi maiuscoli, quanti record è possibile indirizzare?

Risulta:

$$26^5 = 11.881.376 \text{ record}$$

che indica una quantità improponibile; pensando di avere occupati 2.000.000 di record, si avrebbe uno spreco di spazio superiore all'80%.

Collisioni

Escludendo il caso ideale della trasformazione perfetta, restano da esaminare le altre due eventualità:

- $S_I >> S_K$;
- $S_I << S_K$

Nel primo caso c'è ovviamente uno spreco di memoria, nel secondo caso nasce, invece il rischio che due chiavi diverse possano generare lo stesso indirizzo, situazione detta **collisione**, alla quale ci dedichiamo nella prossima Unità. Per ora ci basti sapere che le collisioni portano ad un degrado dei tempi di elaborazione causati dagli accessi supplementari ai dati.

Scelta della funzione *hash*

In pratica, la scelta della funzione *hash* andrà valutata caso per caso, tenendo conto:

- di velocità ed efficienza del calcolo della trasformazione chiave → indirizzo;
- dell'uniformità della distribuzione delle chiavi;
- dell'uniformità della distribuzione degli indirizzi
- di eventuali collisioni possibili
- della riproducibilità, ossia che dovrà dare, a parità di chiave, sempre lo stesso indirizzo;

Tecniche *hash*

Consideriamo allora il caso generale in cui:

- si hanno trasformazioni non perfette;
- si abbiano chiavi alfanumeriche

e presentiamo due tra le numerose tecniche esistenti:

- metodo della divisione
- metodo della moltiplicazione

Indichiamo, nel seguito, con **integer**(K) una generica funzione che converte la chiave *K* da alfanumerico ad intero. In seguito, vedremo anche le varie forme che **integer**(K) può assumere.

Il metodo della divisione

Una semplice funzione *hash* è quella che calcola l'indirizzo come resto della divisione tra la chiave *K*, (convertita in intero), e il numero dei record *N*. Questa si dice **metodo della divisione**, si tratta di un metodo molto veloce espresso dalla relazione seguente

f (K) = **integer**(K) **mod** N

Ad esempio, se dobbiamo memorizzare 47 record in un file, la funzione che individua l'indirizzo del record di chiave *K* è la seguente:

f (K) = **integer**(K) **mod** 47

che genera indirizzi tra 0 e 46.

Il metodo della divisione
- la scelta di N

Sappiamo che in una trasformazione perfetta le chiavi vengono distribuite in modo casuale all'interno del file. Osserviamo gli esempi seguenti:

Ad esempio, se si usasse N = 100, si avrebbe:

K	h(K)
1**23**	23
23**23**	23
156**23**	23

Ad esempio, se si usasse $N = 2^3$, si avrebbe:

K	h(K)
10110**101**	101
111**101**	101
100011101**101**	101

Gli indirizzi ricavati non hanno la stessa distribuzione di probabilità, poiché il valore 23 è ricorrente.

Come si può fare in modo che tutti gli indirizzi siano distribuiti in modo uniforme?

Il metodo della divisione
- la scelta di N

In pratica, si è dimostrato che le chiavi hanno la stessa probabilità di presentarsi solo se N è un numero primo, diverso da una potenza della base.

ATTIVITA': Scegliendo, ad esempio, N = 101, quali sarebbero gli indirizzi generati?

K	h(K)
1**23**	22
23**23**	0
156**23**	69

Il metodo della divisione
- la scelta di N

Se si ha a che fare con un valore di N non primo, si può operare come segue.

1. Individuare il numero primo più vicino ad N, che indichiamo con N_p;
2. Calcolare il R.R.N. rispetto a N_p con:

$$R = \textbf{integer}(K) \textbf{ mod } N_p$$

4. Applicare la proporzione

$$N_p : R = N : RRN$$

da cui

$$RRN = (R * N) / N_p$$

Il metodo della moltiplicazione

Il **metodo di moltiplicazione** per definire le funzioni *hash* opera in tre passi:

- si moltiplica la chiave K per una costante A in [0,1]
- si estrae la parte frazionaria del risultato
- si moltiplica questo valore per N e si prende la parte intera, per difetto

Analiticamente si ha:

$$h(K)= \lfloor N*(K*A \textbf{ mod } 1) \rfloor$$

Ad es. dato N=10000, A=0.6, K=123456, si ha:

$h(K)= \lfloor 10000 * (123456*0.6 \textbf{ mod } 1) \rfloor =$

$= \lfloor 10000 * (76300.0041151.. \textbf{ mod } 1) \rfloor =$

$= \lfloor 10000 * (0.0041151..) \rfloor =$

$= \lfloor 41.151.. \rfloor = 41$

Il metodo della moltiplicazione

Questo metodo presenta i seguenti vantaggi:

- se si sceglie $N = 2^p$ per un qualche p, si semplificano notevolmente i calcoli
- non ci sono valori critici di N

La funzione integer(K)

La funzione **integer**(K), detta **formula** (o **funzione**) **di randomizzazione,** converte la chiave, rappresentata da una stringa, in un numero, per poter ricavare successivamente il R.R.N. in base alla tecnica scelta.

La funzione **integer**(K) deve avere le seguenti caratteristiche:

- dovrebbe essere di semplice computazione, per non appesantire l'elaborazione;
- dovrebbe generare da K un numero molto più grande di N

Funzioni di randomizzazione

Vediamo alcuni esempi di funzione di randomizzazione:

- chiave parziale
- somma codici ASCII
- somma posizioni alfabetiche
- somma delle cifre

Scelte di integer(K)
- chiave parziale

Un primo modo elementare di implementare la funzione **integer**(K) potrebbe essere quello di prendere un sottoinsieme della chiave, o l'unione di più parti di questa, e trasformarlo in un numero intero.

Ad esempio, considerando i primi 4 byte della chiave, si avranno dei *sinonimi* già prima di applicare il calcolo del **mod** (le chiavi "ROSSI", "ROSSINI", "ROSSETTI", "ROSSINIANI" hanno i primi 4 caratteri uguali, pur rappresentando chiavi diverse fra di loro).

Occorrerebbe, per questo, scegliere di formare la chiave numerica prendendo i quattro caratteri che diano il minor numero di sinonimi (per trovarli, di solito, non si può fare altro che utilizzare indagini statistiche).

Scelte di integer(K)
- somma codici ASCII

Un altro caso elementare di funzione **integer**(K) è quello in cui l'indirizzo viene ricavato in base al codice ASCII dei caratteri che compongono la chiave.

Supponendo che la chiave sia composta da 4 caratteri alfabetici, si considera la somma dei codici ASCII e la si divide per il numero N di record presenti nel file, ottenendo il valore di R.R.N. nel file.

In particolare volendo distribuire i record in un hash file con 23 record:

'A' 'n' 'z' 'a'

65 + 110 + 122 + 97 = 394

integer("Anza") = 394 **mod** 23 = 3

Scelte di integer(K)
- somma posizioni alfabetiche

Un altro caso molto semplice di funzione **integer**(K) è quello in cui l'indirizzo viene ricavato come somma delle posizioni alfabetiche di ciascun carattere, da dividere per il numero N di record presenti nel file, ottenendo il valore di R.R.N. nel file.

In particolare volendo applicare la funzione ad una targa, es. "AC113GH" si ha (supponendo N = 103):

'A' 'C' '1' '1' '3' 'G' 'H'

1 + 3 + 1 + 1 + 3 + 7 + 8

integer("AC113GH") = 24 **mod** 103 = 24

Scelte di integer(K)
- somma delle cifre (chiavi numeriche)

Un ultimo caso di funzione **integer**(K), specifico per chiavi numeriche, è quello in cui l'indirizzo viene ricavato come somma delle cifre della chiave, da dividere per il numero N di record presenti nel file, ottenendo il valore di R.R.N. nel file.

In particolare se K = "123456789" si ha (supponendo N = 30):

integer("123456789") = 45 **mod** 30 = 15

Tabelle *hash*

I dati in memoria centrale sotto forma di tabella, possono essere acceduti usando:

- la chiave come indice, se la memoria è sufficiente; un array ordinario può essere utilizzato è possibile una posizione per ogni possibile chiave.
- una tabella *hash*, quando il numero delle chiavi è piccolo rispetto al numero totale di possibili chiavi;
- una lista per memorizzare le sole chiavi effettivamente presenti

Le **tabelle hash** sono strutture dati che consentono una veloce gestione (ricerca e aggiornamento) dei dati in memoria centrale, ottimizzando spazio occupato e tempo di esecuzione.

Argomenti

- Organizzazione *hash*
- La funzione *hash*
- Trasformazione perfetta
- Collisioni
- Scelta della funzione *hash*
- Tecniche *hash*
- Il metodo della divisione
  - la scelta di N
- Il metodo della moltiplicazione
- La funzione **integer(K)**
- Funzioni di randomizzazione
  - chiave parziale
  - somma codici ASCII
  - somma posizioni alfabetiche
  - somma delle cifre

Altre fonti di informazione

- A.Lorenzi-D.Rossi, Le basi di dati e il linguaggio SQL – ed. ATLAS
- G.Callegarin – Corso di Informatica Generale 3 – CEDAM
- P.Gallo, F.Salerno – Informatica 2 – ed. Minerva Italica