Organizzazione Fisica dei Dati (Parte II)

Angelo Montanari

Dipartimento di Matematica e Informatica
Università di Udine

Sommario

Introduzione alle strutture ad indice per file

Indici ordinati di singolo livello:

- indici primari
- indici di clustering
- indici secondari

Indici multilivello (statici)

Strutture ad indice per file

Gli **indici** (index) sono strutture di accesso ai dati utilizzate per rendere più veloce il recupero di record in presenza di determinate condizioni di ricerca.

Alcuni tipi di indici (secondary access path) non influenzano la disposizione fisica dei record sul disco, ma definiscono dei cammini di ricerca alternativi per localizzare efficientemente i record sulla base di opportuni campi indice (indexing field).

I più importanti tipi di indice sono basati su file ordinati (indici di singolo livello) e su strutture dati ad albero (indici multilivello statici, B-alberi, B^+-alberi).

Gli indici possono essere costruiti sulla base di strutture dati di tipo hashing o di altro tipo.

Indici ordinati di singolo livello

L'idea alla base di una struttura ordinata ad indici è simile a quella degli indici per parole che si trovano alla fine dei libri di testo: accanto ad ogni parola vi è la lista dei numeri di pagina dove la parola compare nel testo (alternativamente, bisogna scorrere tutto il testo parola per parola: linear search).

Una struttura ad indici è usualmente definita su un singolo campo di un file (**indexing field**). Ogni indice memorizza per ogni valore del campo una lista di puntatori a blocchi su disco che contengono record con quel valore di campo.

Gli indici sono ordinati in modo da consentire una **ricerca binaria**. Il file indice è molto più piccolo del file di dati: una ricerca binaria sul file indice è molto efficiente.

Diversi tipi di indice (di singolo livello) - 1

Indici primari. Un indice primario è un indice specificato rispetto al campo chiave di un file di record fisicamente ordinato rispetto a tale campo (ordering key field).

Si noti che:

1. tale campo è utilizzato per ordinare fisicamente i record del file sul disco (**ordering** key field);
2. ogni record è identificato univocamente dal valore che assume su tale campo (ordering **key** field).

Diversi tipi di indice (di singolo livello) - 2

Indice di clustering. L'indice di clustering si usa nel caso in cui l'ordering field non sia un key field (più record nel file possono assumere lo stesso valore rispetto all'ordering field).

Osservazione. Poiché un file può essere ordinato rispetto ad al più un **physical ordering field**, esso può avere o un indice primario o un indice clustering, ma non entrambi.

Indice secondario. Un indice secondario può essere definito su ogni campo (non utilizzato per ordinare il file). Un file può avere più indici secondari.

Indice primario - 1

Un **indice primario** è un file ordinato i cui record, di lunghezza prefissata, possiedono due campi:

$$(\langle \text{chiave primaria} \rangle, \langle \text{indirizzo blocco} \rangle)$$

Nel file indice esiste una index entry (o index record) i per ogni blocco b del file di dati:

$$\langle k(i), p(i) \rangle$$

dove $k(i)$ è un valore minore (o uguale) a quello di tutte le chiavi dei record in b, ma maggiore del valore di tutte le chiavi del blocco che precede b, e $p(i)$ è l'indirizzo del blocco b.

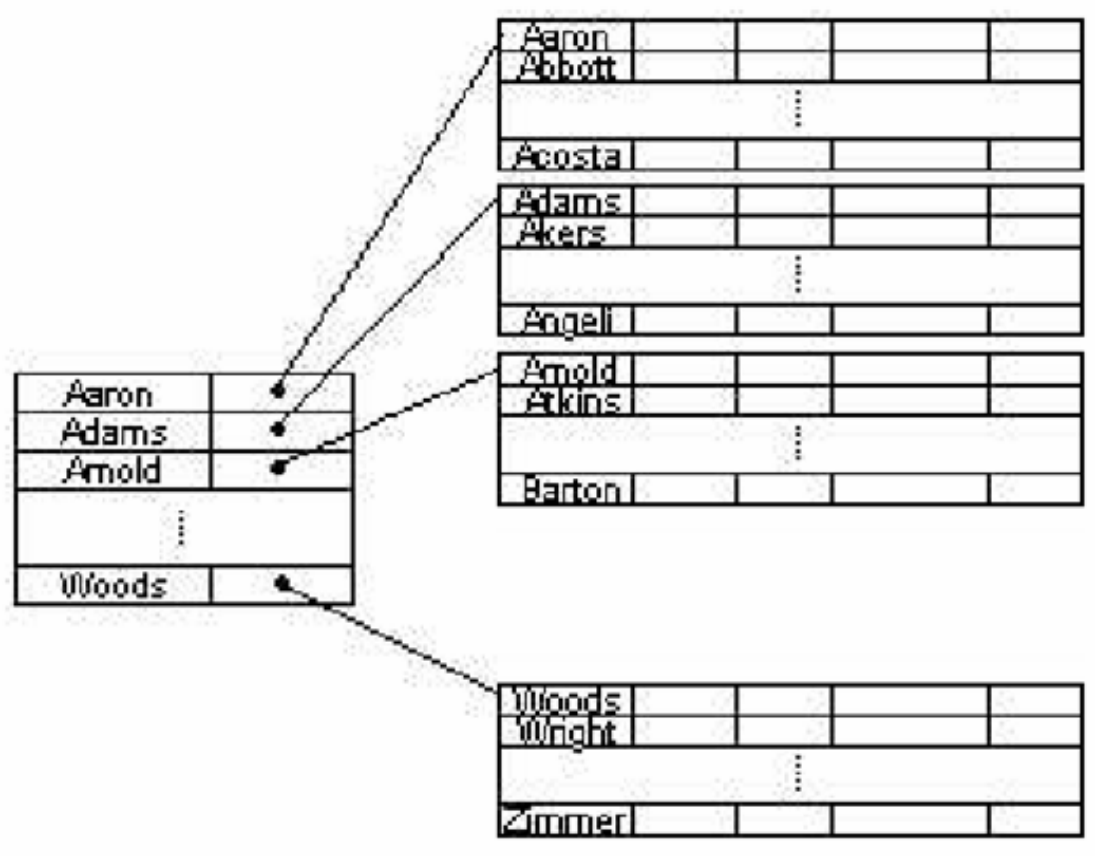
Indice primario - 2

Spesso si assegna a $k(i)$ il valore minimo delle chiavi presenti in b (valore della chiave del primo record del blocco). Se b è il primo blocco, conviene scegliere $k(i) = -\infty$, dove $-\infty$ rappresenta un valore inferiore a quello di ogni possibile chiave.

Si noti che (i) il primo campo del file indice è una **chiave** per tale file e (ii) il file indice è **ordinato** rispetto ad essa.

Esempio. Supponiamo di avere un file di dati che rappresenti le tuple della relazione IMPIEGATO(NOME, DATAN,LAVORO, STIPENDIO, SESSO) di una base di dati AZIENDA. Assumiamo che NOME sia la chiave primaria. Tale file ed il corrispondente file indice sono rappresentati nella figura riportata nel lucido successivo (versione inglese).

Indice primario - 3



Indice primario - 4

Il primo record di ogni blocco del file di dati è definito anchor record del blocco (**block anchor**). Un indice primario è un indice **non denso** in quanto include una entry per ogni blocco (un indice è detto denso se contiene una entry per ogni record del file).

Un record con valore della chiave primaria uguale a k si trova nel blocco di indirizzo $p(i)$, con

$$k(i) \leq k < k(i+1)$$

Per recuperare un record, noto il valore k della sua chiave primaria, occorre effettuare una ricerca binaria sul file indice per cercare l'appropriata index entry i e quindi recuperare il blocco il cui indirizzo è $p(i)$.

Indice primario - 5

Esempio. Supponiamo di avere un file ordinato contenente $r = 30000$ record memorizzati su un disco con dimensioni del blocco $B = 1024$ byte. I record siano di lunghezza fissa $R = 100$ byte e unspanned. Il blocking factor per il file è:

$$bfr = \lfloor B/R \rfloor = \lfloor 1024/100 \rfloor = 10 \text{ record per blocco}$$

Il numero di blocchi necessari per memorizzare il file è:

$$n_b = \lceil r/bfr \rceil = \lceil 30000/10 \rceil = 3000 \text{ blocchi}$$

Una ricerca binaria sul file di dati richiede approssimativamente:

$$log_2 nb = log_2 3000 = 12 \text{ accessi a blocco}$$

Indice primario - 6

Esempio (continua). Supponiamo ora che il campo contenente la ordering key del file sia lungo $V = 9$ byte e che il puntatore ad un blocco sia lungo $P = 6$ byte. La dimensione di ogni index entry R_i è:

$$R_i = 9 + 6 = 15 \text{ byte},$$

mentre il blocking factor è:

$$bfr_i = \lfloor B/R_i \rfloor = \lfloor 1024/15 \rfloor = 68 \text{ index entry per blocco}$$

Il numero totale di index entry r_i è uguale al numero di blocchi del file, che è 3000. Il numero di blocchi necessario per l'indice è:

$$nb_i = \lceil r_i/bfr_i \rceil = \lceil 3000/68 \rceil = 45 \text{ blocchi}$$

Una ricerca binaria sul file indice richiede:

$$\lceil log_2 nb_i \rceil = \lceil log_2 45 \rceil = 6 \text{ accessi a blocco}$$

Indice primario - 7

Conclusione: per cercare un record utilizzando l'indice occorrono 6 accessi a blocchi dell'indice più 1 accesso al blocco del file contenente il record, per un totale di $6 + 1 = 7$ accessi, contro i 12 accessi richiesti da una ricerca binaria sul file dei dati (senza uso dell'indice).

Il **problema** principale degli indici primari (analogamente a quanto accadeva con i file ordinati) è quello dell'inserimento e della cancellazione di record, con in più l'onere di dover modificare anche le index entry. Per ovviare a tale problema, si può utilizzare un file di overflow non ordinato (come visto nel capitolo sull'hashing), oppure aggiungere una lista di record di overflow ad ogni blocco del file di dati.

Indici di clustering - 1

Se i record di un file sono ordinati fisicamente rispetto ad un campo non chiave (record diversi possono assumere lo stesso valore su tale campo), tale campo è chiamato **clustering field**.

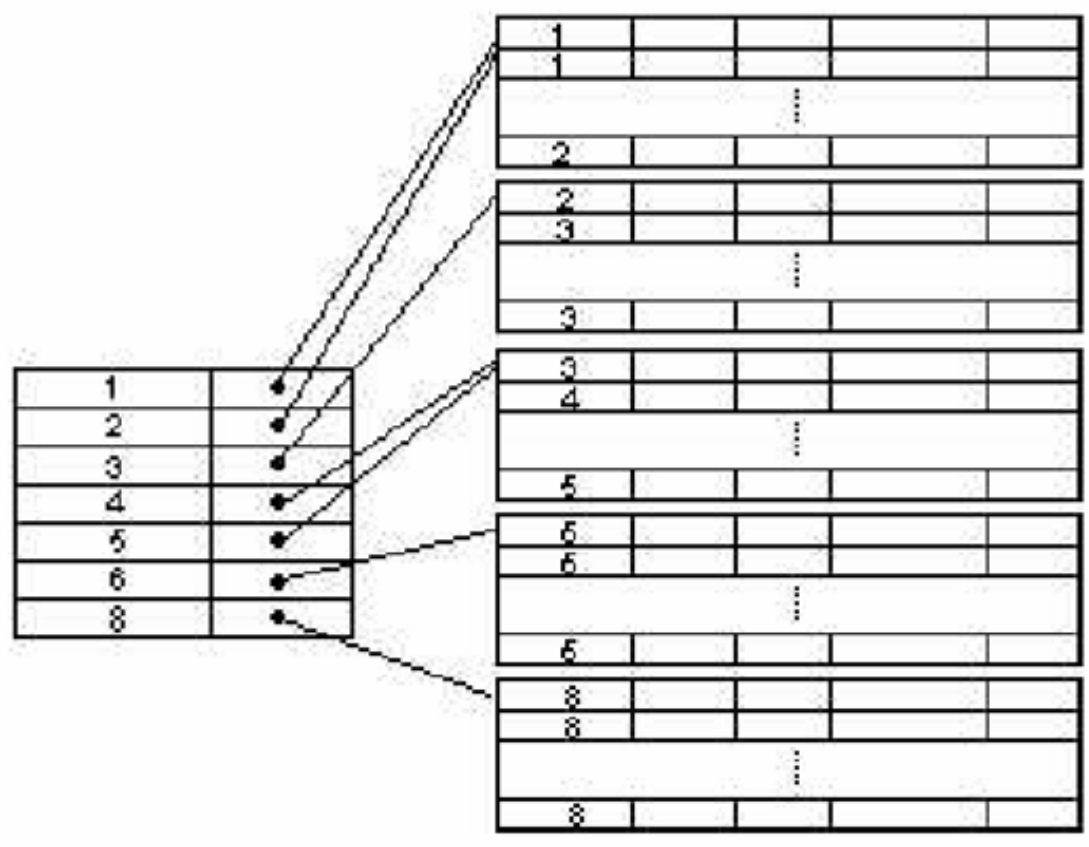
In tal caso, è possibile creare un diverso tipo di indice (**indice di clustering**) per rendere più efficiente il recupero di record che assumono lo stesso valore sul clustering field.

Un clustering index è un file ordinato con due campi:

$$(\langle \text{clustering field} \rangle, \langle \text{indirizzo blocco} \rangle)$$

Esiste una entry nel clustering index per ogni valore distinto del clustering field, contenente il valore ed un puntatore al primo blocco nel file dei dati che ha un record con quel valore per il suo clustering field.

Indici di clustering - 2



Indici di clustering - 3

Problemi con inserimenti e cancellazioni. L' inserimento e la cancellazione di record creano dei problemi anche in questo caso, in quanto i record di dati sono fisicamente ordinati.

Per ridurre il problema nel caso di **inserimenti**, si riserva un intero blocco per ogni valore del clustering field; tutti i record con quel valore sono posizionati nel relativo blocco ed eventualmente in blocchi addizionali ad esso collegati.

Indici secondari - 1

Un indice secondario è (ancora) un file ordinato con due campi:

$$(\langle \text{indexing field} \rangle, \langle \text{indirizzo blocco} \rangle)$$

A differenza dei due tipi di indici precedenti, è possibile associare più indici secondari (e quindi indexing field) ad uno stesso file.

Consideriamo una struttura con accesso attraverso **indici secondari** su un **campo chiave** (campo che assume un valore diverso su ogni record del file di dati - secondary key).

Vi sarà una index entry per ogni record del file di dati, contenente il valore della chiave secondaria per il record ed un puntatore al blocco in cui il record è memorizzato (puntatore al blocco anziché al record..).

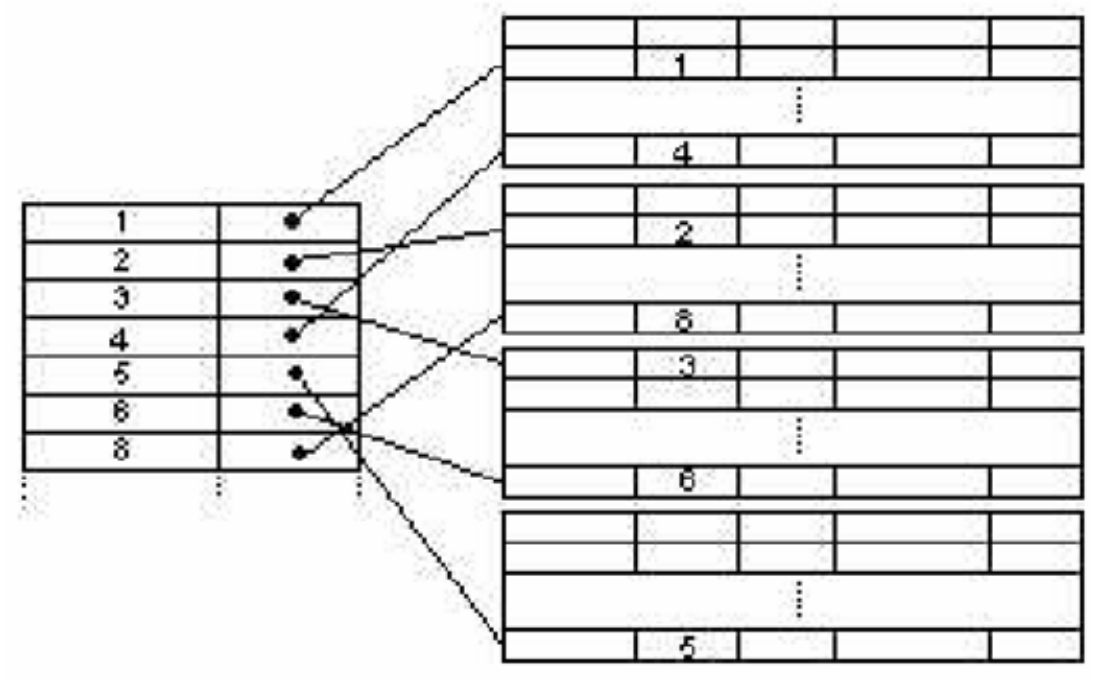
Indici secondari - 2

Un indice secondario su un key field è un indice **denso** (a differenza degli indici primari, gli indici secondari contengono una entry per ogni record, e non per ogni blocco, del file di dati): poiché i record del file di dati non sono fisicamente ordinati rispetto al valore del secondary key field, non è possibile utilizzare i block anchor.

Quando un blocco è trasferito in memoria primaria, bisogna effettuare una ricerca del record desiderato.

Un indice secondario richiede molto più spazio ed un tempo di ricerca maggiore di un indice primario in ragione dell'elevato numero di entry (indice denso). Tuttavia, la riduzione del tempo di ricerca di un record sulla base del valore da esso assunto sull'indexing field è molto significativa (in questi casi, disponendo del solo indice primario, occorre eseguire una scansione lineare).

Indici secondari - 3



Attenzione: si tratta in ogni caso di puntatori a blocco, non a record (è computazionalmente più vantaggioso)

Indici secondari - 4

Esempio. Supponiamo di avere un file ordinato con $r = 30000$ record, memorizzati su un disco con dimensione del blocco pari a $B = 1024$ byte. I record siano di lunghezza fissa $R = 100$ byte e unspanned. Come visto in precedenza, il blocking factor è $bfr = 10$ record per blocco, e il numero di blocchi necessari per memorizzare il file è $n_b = 3000$.

Una ricerca lineare sul file di dati richiede mediamente

$$n_b/2 = 3000/2 = 1500 \text{ accessi a blocco}$$

Come in precedenza, assumiamo che la dimensione di ogni index entry R_i sia uguale a $9 + 6 = 15$ byte ed il blocking factor bfr_i sia di 68 entry per blocco.

Indici secondari - 5

Esempio (continua). Nel caso di indice denso, il numero totale di index entry r_i è uguale al numero di record del file, che è 30000. Ne consegue che il numero di blocchi necessario per l'indice è:

$$nb_i = \lceil ri/bfr_i \rceil = \lceil 30000/68 \rceil = 442 \text{ blocchi,}$$

contro i 45 blocchi necessari nel caso di indice primario.

Una ricerca binaria sul file indice richiede:

$$\lceil log_2 nb_i \rceil = \lceil log_2 442 \rceil = 9 \text{ accessi a blocco.}$$

Per cercare un record utilizzando l'indice (secondario), occorre 1 accesso addizionale ad un blocco del file per un totale di $9 + 1 = 10$ accessi, contro i 1500 accessi necessari in media per una ricerca lineare (poco più dei 7 accessi richiesti dalle ricerche che sfruttano l'indice primario).

Indici secondari - 6

E' possibile creare un indice secondario anche su un **campo non chiave** (nonkey field) del file. In tal caso, numerosi record possono assumere lo stesso valore sull'indexing field.

Esistono diverse possibili modalità di implementazione di tale indice:

(1) includere più index entry con lo stesso valore di $k(i)$, una per ogni record (indice denso);

(2) considerare record di lunghezza variabile per le index entry, con un campo (più volte) ripetuto per il puntatore. Viene mantenuta una lista di puntatori:

$$\langle p(i,1), .., p(i,k) \rangle$$

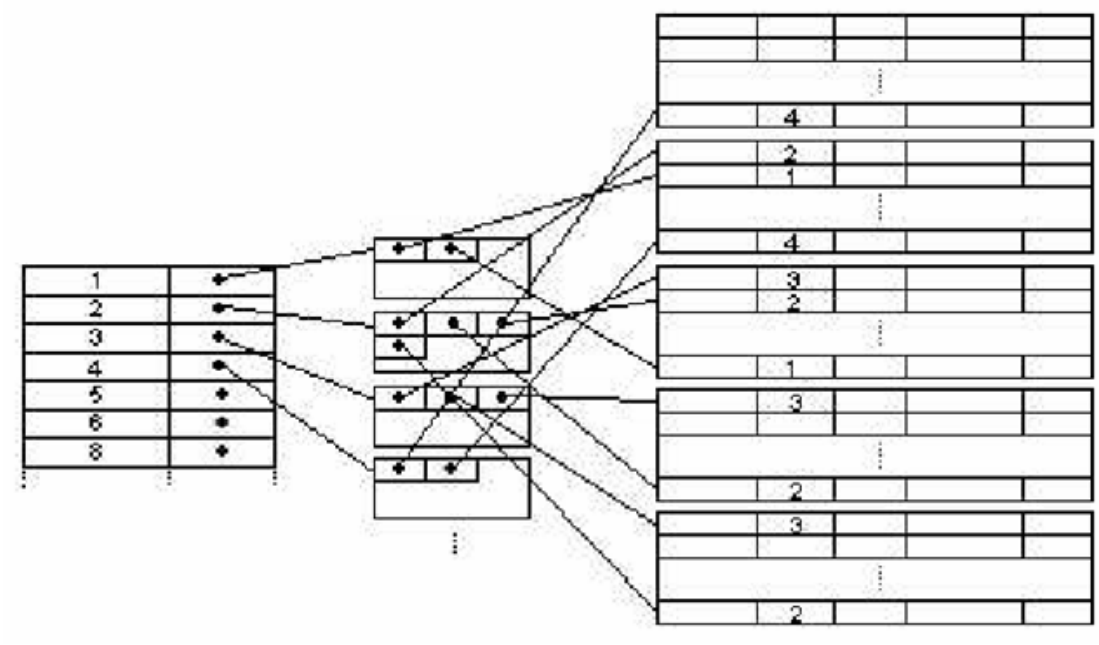
nella index entry per $k(i)$ (un puntatore ad ogni blocco che contiene un record il cui indexing field value è uguale a $k(i)$).

Indici secondari - 7

(3) Il puntatore $p(i)$ in una index entry $\langle k(i), p(i)\rangle$ punta ad un blocco; ogni puntatore presente nel blocco punta ad (un blocco contenente) un record del file di dati con valore $k(i)$ dell'indexing field.

Se i valori di $k(i)$ sono condivisi da molti record, e, quindi, i corrispondenti puntatori ai (blocchi contenenti i) record non possono essere contenuti in un unico blocco, si utilizza una lista di blocchi (vedi figura riportata nel lucido successivo).

Indici secondari - 8



Indice secondario costruito su un campo non chiave del file (caso 3).

Indici multilivello - 1

Una ricerca binaria richiede approssimativamente $log_2 b_i$ accessi a blocco per un indice con b_i blocchi, poiché ogni passo dell'algoritmo riduce la parte del file indice da analizzare di un fattore 2.

L'idea di utilizzare **indici multilivello** è nata dalla necessità di ridurre ad ogni passo la parte dell'indice che rimane da analizzare di un fattore pari a bfr_i (che è abitualmente > 2). Il valore bfr_i è detto **fan-out** (fo) dell'indice multilivello.

La ricerca su un indice multilivello richiede approssimativamente $log_{fo} b_i$ accessi a blocco.

Indici multilivello - 2

Un indice multilivello considera il file indice (primo livello) come un file ordinato con un valore distinto per ogni $k(i)$ e crea un indice primario per il primo livello (**secondo livello**).

Poiché il secondo livello è un indice primario, si possono in realtà usare dei block anchor in modo che il secondo livello abbia una entry per ogni blocco (e non per ogni entry) del primo livello.

Il bfr_i per il secondo livello (e per tutti i livelli successivi) è lo stesso dell'indice del primo livello, poiché tutte le index entry hanno la stessa dimensione (ognuna ha un $\langle field\ value \rangle$ e un $\langle indirizzo\ blocco \rangle$).

Indici multilivello - 3

Se il primo livello ha r_1 entry e $bfr_i = fo$, allora il primo livello ha bisogno di $\lceil r_1/fo \rceil$ blocchi, che è il numero di entry r_2 necessarie al secondo livello dell'indice.

Si può ripetere questo processo per il secondo livello. Il **terzo livello**, che è un indice primario per il secondo livello, ha una entry per ogni blocco di secondo livello.

Il numero di entry al terzo livello è:

$$r_3 = \lceil r_2/fo \rceil$$

Indici multilivello - 4

Osservazione. E' richiesto un secondo livello solo se il primo livello richiede più di un blocco di spazio su disco; è richiesto un terzo livello solo se il secondo richiede più di un blocco, e così via.

Si ripete il procedimento fino a quando tutte le entry di un certo index level t (**top index level**) stanno in un singolo blocco.

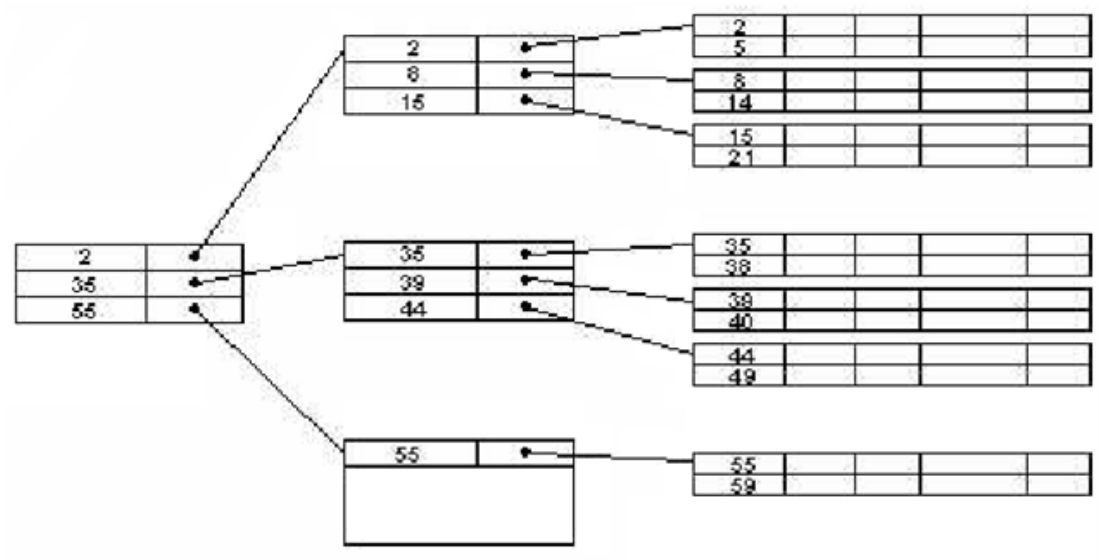
Ogni livello riduce il numero di entry del livello precedente di un fattore fo. Il top index level t si può calcolare a partire dal vincolo:

$$1 \geq r_1 / fo^t$$

Un indice multilivello con r_1 entry al primo livello avrà approssimativamente t livelli, dove

$$t = \lceil log_{fo} r_1 \rceil$$

Indici multilivello - 5



Indici multilivello - 6

Esempio. Supponiamo di avere un file non ordinato con $r = 30000$ record, memorizzati su un disco con dimensione del blocco pari a $B = 1024$ byte. I record siano di lunghezza fissa $R = 100$ byte e unspanned.

Come visto in precedenza, il blocking factor è $bfr = 10$ record per blocco e il numero di blocchi necessari per memorizzare il file è $n_b = 3000$.

Come in precedenza, assumiamo che la dimensione di ogni index entry R_i sia uguale a $9 + 6 = 15$ byte ed il blocking factor bfr_i sia di 68 entry per blocco.

Indici multilivello - 7

Vogliamo ora convertire tale file con un indice secondario denso in un **indice multilivello**. Il blocking factor bfr_i è anche il fan-out dell'indice multilivello.

Il numero di blocchi di primo livello è $b_1 = 442$, mentre quello dei blocchi di secondo livello è:

$$b_2 = \lceil b_1/fo \rceil = \lceil 442/68 \rceil = 7 \text{ blocchi}$$

e quello del terzo livello è:

$$b_3 = \lceil b_2/fo \rceil = \lceil 7/68 \rceil = 1 \text{ blocco}$$

Indici multilivello - 8

Il terzo livello è il top level dell'indice e $t = 3$.

Accesso con indice secondario:

$$\lceil log_2 b_1 \rceil = \lceil log_2 442 \rceil = 9 \text{ accessi a blocco}$$

più un accesso a un blocco del file di dati.

Per accedere ad un record utilizzando l'indice multilivello, bisogna accedere ad un blocco per ogni livello più l'accesso al blocco del file di dati. Ne segue che il numero totale di accessi a blocchi è pari a:

$$t + 1 = 3 + 1 = 4$$

Indici multilivello - 9

Forniamo ora una procedura di ricerca di un record in un file di dati che usa un indice primario multilivello non denso con t livelli. Sia $\langle k_j(i), p(i)\rangle$ una entry i al livello j. Si vuole cercare un record la cui chiave primaria è k. Eventuali record di overflow sono ignorati.

$p :=$ address of top level block of index;
for $j = t$ **step** -1 **to** 1 **do**
begin
read the index block (at $j-th$ index level) whose address is p;
search block p for entry i such that $K_j(i) \leq K < K_j(i+1)$(if K_j is the last entry in the block, it suffices to satisfy $K_j(i) \leq K$);
$p := P_j(i)$ # pick appropriate pointer at $j-th$ index level
end;
read the data file block whose address is p;
search block p for record with key $= K$;