

Lezione 23– File con indice: caratteristiche ed esercizi

Prof.ssa Maria De Marsico
demarsico@di.uniroma1.it



SAPIENZA
UNIVERSITÀ DI ROMA

Dati che ammettono un ordinamento significativo per l'applicazione



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Quando le chiavi ammettono un ordinamento **significativo** per l'applicazione, è più conveniente utilizzare un'organizzazione fisica dei dati che ne tenga conto
- Interi e stringhe ammettono i consueti ordinamenti (lessicografico per le stringhe)
- Per campi multipli, si ordina sul primo campo, poi sul secondo e così via (in pratica un'estensione dell'ordine lessicografico in cui i simboli sono i valori dei campi)

•
Un *alfabeto finito totalmente ordinato* di simboli è un insieme
 $\Sigma = (\delta_1, \delta_2, \dots, \delta_n), \dots$ dotato di un ordine totale . $\delta_1 < \delta_2 < \dots < \delta_n$

Date due sequenze di simboli

$$I = \delta_{i1} \delta_{i2} \dots \delta_{in}$$
$$J = \delta_{j1} \delta_{j2} \dots \delta_{jm}$$

diciamo che $I < J$ se esiste un numero $k \in \mathbb{N}$ per cui
 $\delta_{i1} \delta_{i2} \dots \delta_{ik} = \delta_{j1} \delta_{j2} \dots \delta_{jk}$

e vale una delle seguenti relazioni:

$$\delta_{i(k+1)} < \delta_{j(k+1)} \quad \text{oppure} \quad n = k < m$$

•

- Algoritmo di confronto (banale ...)
- La regola data sopra è equivalente al seguente algoritmo di confronto:
- si pone $n=1$
- si **confrontano** i simboli nella **posizione n-esima** della stringa:
 - se una delle due stringhe **non possiede** l'elemento n-esimo, **allora è minore** dell'altra e l'algoritmo termina
 - se entrambe le stringhe **non possiedono** l'elemento n-esimo, allora sono **uguali** e l'algoritmo termina
 - se i simboli **sono uguali**, si passa alla posizione successiva della stringa ($n=n+1$)
 - se questi sono **diversi**, il loro **ordine** è l'ordine delle stringhe



Primo esempio significativo

File **ISAM**

(Indexed Sequential Access Method)

File con indice

	matr	cognome	...
blocco 1	003	Neroni	...
	005	Verdi	...
	009	Carini	...
	010	Rossi	...

blocco 2	031	Bianchi	...
	048	Bellini	...
	050
	099

blocco 3	101
	123
	124
	133

...
blocco n	220
	234

Il file viene ordinato
in base al valore della
chiave di ricerca

In genere viene lasciata
una certa percentuale
di spazio libero
in ogni blocco

Viene creato un nuovo file:
il **file indice** che
contiene un **record**
per ogni **blocco** del
file principale

ogni record **del file**
indice ha **due campi** che
contengono:
un puntatore ad un
blocco del
file principale
e il **più piccolo valore**
della chiave presente
nel blocco

blocco 1

Indice	
matr	
-∞	●
031	●
101	●
...	

File principale		
matr	cognome	...
003	Neroni	...
005	Verdi	...
009	Carini	...
010	Rossi	...

blocco 2

...	...
...	...
...	...
...	...

031	Bianchi	...
048	Bellini	...
050
099

blocco 3

...	...
...	...
...	...
...	...

101
123
124
133

...

blocco m

...	...
220	●

220
234

Ricerca

Esempi:

-Il record con chiave **090** deve trovarsi nel blocco del file principale che contiene **031** in quanto i valori della chiave nei blocchi precedenti sono <031 e quelli nei blocchi successivi sono ≥ 101 (**031** \leq 090 $<$ 101)

-Il record con chiave **234** deve trovarsi nell'ultimo blocco del file principale in quanto i valori della chiave nei blocchi precedenti sono <220 (**220** \leq 234)

Indice

matr

blocco 1

$-\infty$	
031	
101	
...	...

blocco 2

...	...
...	...
...	...
...	...

blocco 3

...	...
...	...
...	...
...	...

...

blocco m

...	...
220	

090

234

blocco 1

Indice

matr

-∞	•
031	•
101	•
...	

blocco 2

...	...
...	...
...	...
...	...

blocco 3

...	...
...	...
...	...
...	...

...

blocco m

...	...
220	•

File principale

matr

cognome

...

003	Neroni	...
005	Verdi	...
009	Carini	...
010	Rossi	...

031	Bianchi	...
048	Bellini	...
050
099

101
123
124
133

...
220
234

Ricerca

Per ricercare un **record** con valore della **chiave k** occorre ricercare sul file indice un valore **k' della chiave** che **ricopre k**, cioè tale che:

- **$k' \leq k$** e
- se il record con chiave k' **non è** l'ultimo record del file indice e k'' è il valore della chiave nel record **successivo $k < k''$**

Indice

matr

blocco 1

$-\infty$	
031	
101	
...	...

blocco 2

...	...
...	...
...	...
...	...

blocco 3

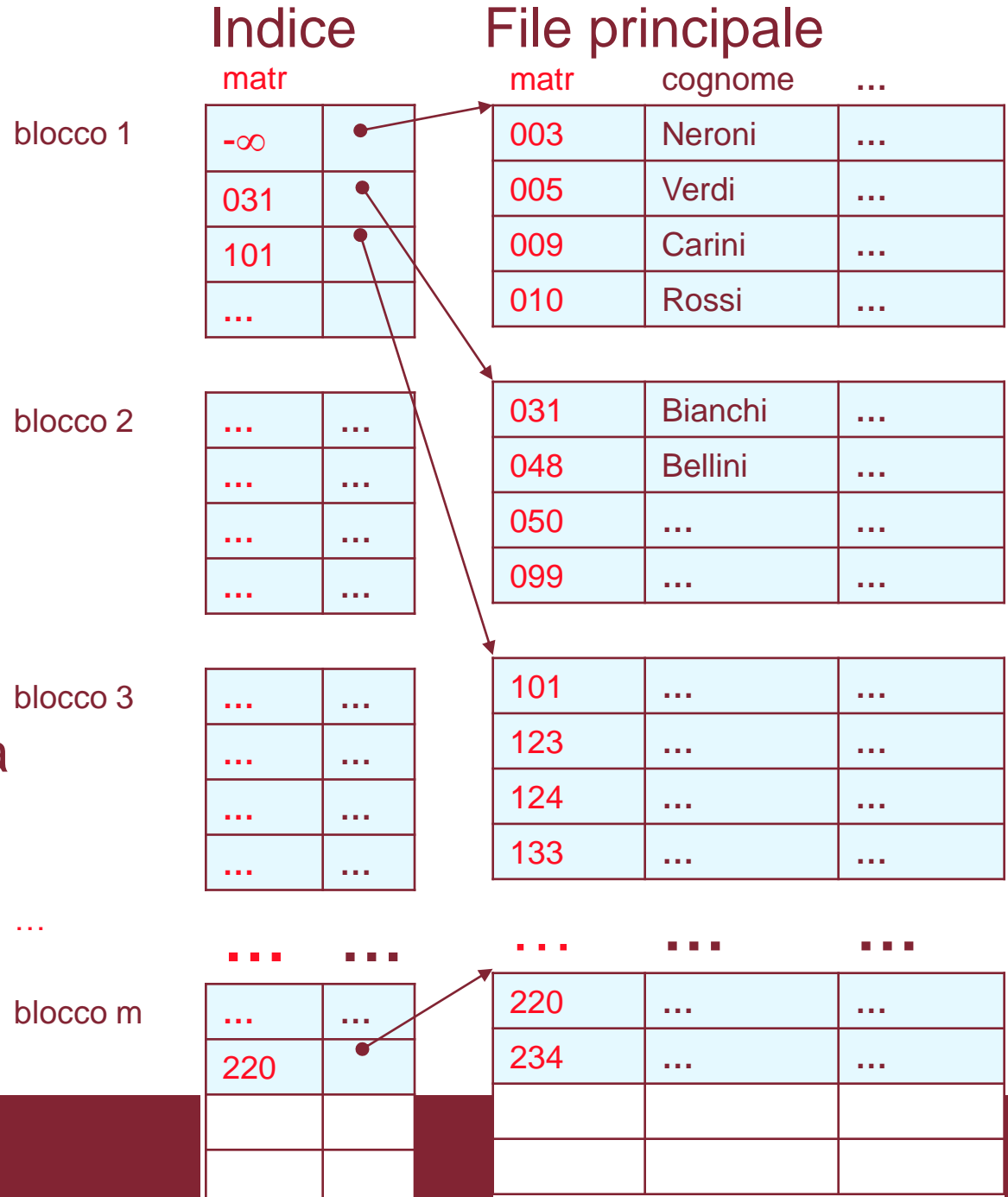
...	...
...	...
...	...
...	...

...

blocco m

...	...
220	

La **ricerca** di un record con chiave k richiede una **ricerca sul file indice** + **1 accesso** in lettura sul file principale





Poichè il file indice è ordinato in base al valore della chiave, la ricerca di un valore che ricopre la chiave può essere fatta in modo efficiente mediante la **ricerca binaria**.

Ricerca binaria

Ricerca binaria

Si fa un accesso in lettura al **blocco** $(m/2)+1$ e si confronta **k con k1 (prima chiave del blocco)**.

Se $k=k_1$ abbiamo finito

Se $k < k_1$ allora si **ripete** il procedimento sui blocchi **da 1 a $(m/2)$**

altrimenti si ripete il procedimento

sui blocchi da **$(m/2)+1$ ad m**

(il blocco $(m/2)+1$ va riconsiderato perché abbiamo controllato solo la prima chiave ...)

Ci si ferma quando lo spazio di ricerca è ridotto ad un unico blocco, quindi dopo $\lceil \log_2 m \rceil$ accessi.

blocco 1

matr	
$-\infty$	
031	
101	
...	...

...

blocco $m/2$

...	...
...	...
...	...
...	...

blocco $(m/2)+1$

k1	...
...	...
...	...
...	...

...

blocco m

...	...
220	

La *ricerca per interpolazione* è basata sulla conoscenza della **distribuzione** dei valori della chiave:

deve essere disponibile **una funzione f** che dati tre valori **$k1$, $k2$, $k3$** della chiave fornisce un valore **che è la frazione dell'intervallo di valori della chiave compresi tra $k2$ e $k3$** in cui deve trovarsi **$k1$** cioè la chiave che stiamo cercando (nella ricerca binaria questa frazione è sempre $\frac{1}{2}$)

Esempio: quando cerchiamo in un elenco telefonico **non** partiamo **sempre** da **metà** ...

- k_1 deve essere confrontato con il valore k della chiave **nel primo record del blocco i** (del file **indice**), dove $i = f(k_1, k_2, k_3) * m$; analogamente a quanto accade nella ricerca binaria.
- Se k_1 è minore di tale valore allora il procedimento deve essere ripetuto sui blocchi **1, 2, ..., i-1**, mentre se è **maggiore** il procedimento deve essere ripetuto sui blocchi **$i, i+1, ..., m$** , finchè la ricerca si restringe ad un unico blocco.

• la ricerca per interpolazione richiede circa $1 + \log_2 \log_2 m$ accessi = **MOLTO** più veloce ma ... **MOLTO** difficile conoscere f e poi la distribuzione dei dati potrebbe cambiare nel tempo

blocco 1

matr

k_2	
...	
...	
...	...

...

blocco i-1

...

...	...
...	...
...	...
...	...

blocco i

k	...
...	...
...	...
...	...

...

blocco m

...

...	...
k_3	

Inserimento

048	Bellini	
-----	---------	--

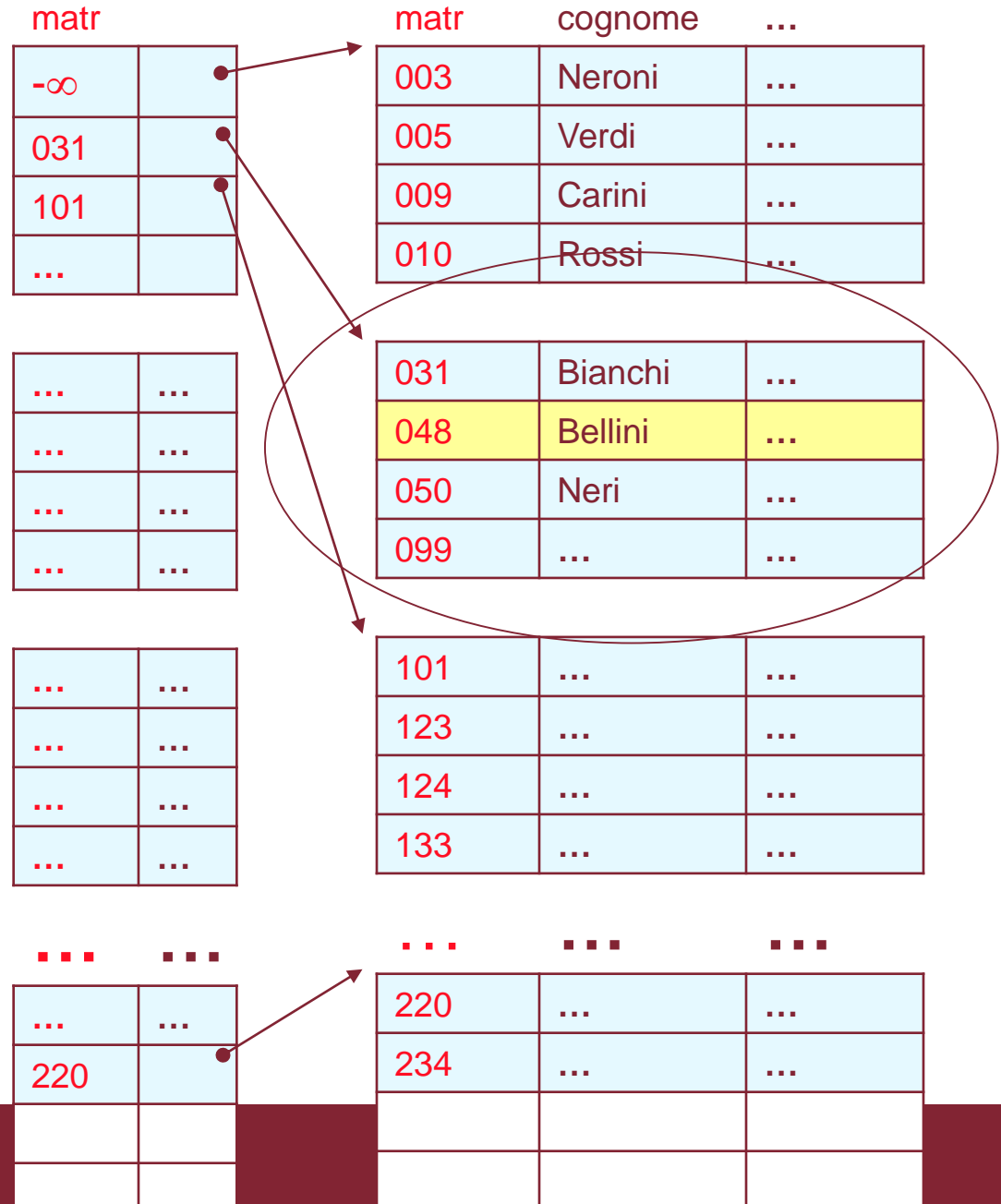
costo per la ricerca ...

The diagram illustrates the concept of a 'matr' (matrix) in a database context. It shows a table with columns 'matr', 'cognome', and '...' (representing other columns). The 'matr' column contains values like $-\infty$, 031, 101, and Arrows point from these values to corresponding rows in a larger table. For example, 031 points to a row with 'Bianchi' and '...', 101 points to a row with 'Neri' and '...', and 133 points to a row with 'Gialli' and '...'.

Inserimento

...+1

(per scrivere
il blocco modificato)
se nel blocco c'è spazio
per inserire
il nuovo record.
Altrimenti ...



Inserimento

048	Bellini	
-----	---------	--

... se c'è spazio
nel blocco successivo

The diagram illustrates the iterative process of finding the top-k results in a matrix-vector multiplication. It shows a sequence of matrices and vectors, with arrows indicating the flow of data and the selection of the next iteration's matrix.

Iteration 1:

- matr:** A matrix with rows $-\infty$, 031, 101, and
- cognome:** A vector with values 003, 005, 009, 010, 031, 050, 099, 100, 101, 123, 124, and

Arrows indicate that the values 031, 101, and ... from the **matr** matrix are used to select the corresponding rows from the **cognome** vector, resulting in the values 031, 050, 099, and 100.

Iteration 2:

- matr:** A matrix with rows ..., ..., ..., and
- cognome:** A vector with values 101, 123, 124, and

Arrows indicate that the values ..., ..., ..., and ... from the **matr** matrix are used to select the corresponding rows from the **cognome** vector, resulting in the values 220 and 234.




Iteration 3:

- matr:** A matrix with rows ..., ..., 220, and
- cognome:** A vector with values 220, 234, and

Arrows indicate that the values ..., ..., 220, and ... from the **matr** matrix are used to select the corresponding rows from the **cognome** vector, resulting in the values 220 and 234.

Inserimento









... possono essere necessari ulteriori accessi

matr	
$-\infty$	
031	
100	
...	

matr	cognome	...
003	Neroni	...
005	Verdi	...
009	Carini	...
010	Rossi	...

<div> <div> <div></div> <div></div> <div></div> </div> </div>	<div> <div> <div></div> <div></div> <div></div> </div> </div>
<div> <div> <div></div> <div></div> <div></div> </div> </div>	<div> <div> <div></div> <div></div> <div></div> </div> </div>
<div> <div> <div></div> <div></div> <div></div> </div> </div>	<div> <div> <div></div> <div></div> <div></div> </div> </div>
<div> <div> <div></div> <div></div> <div></div> </div> </div>	<div> <div> <div></div> <div></div> <div></div> </div> </div>

031	Bianchi	...
048	Bellini	...
050	Neri	...
099	...	

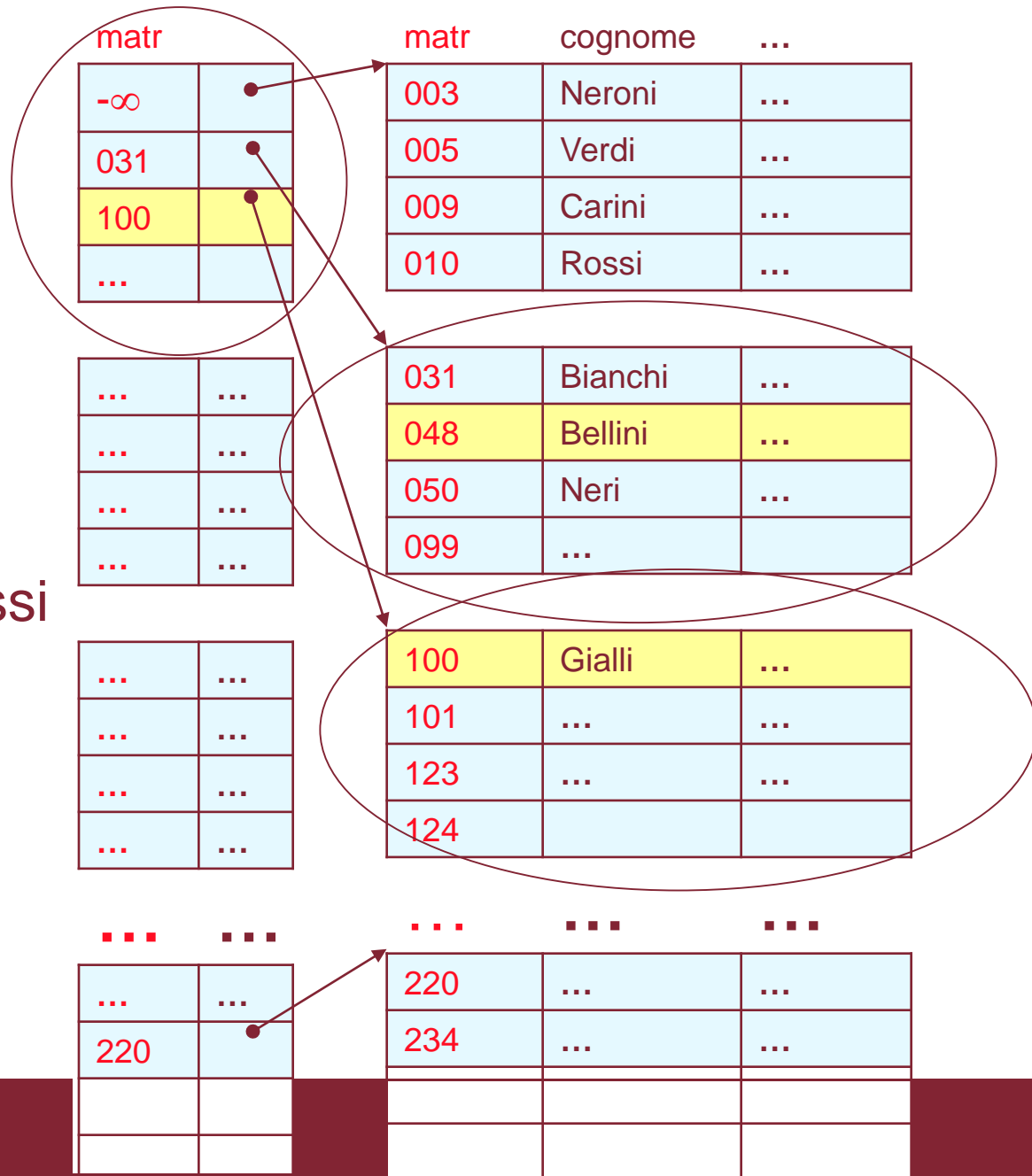
100	Gialli	...
101
123
124		

...	...
220	...

...
220
234

Inserimento

... possono essere
necessari ulteriori accessi



Inserimento

040	Toni	
-----	------	--

Se non c'è spazio né nel blocco precedente né in quello successivo ...

The diagram illustrates the construction of a sparse matrix from a graph. The graph consists of nodes (red squares) and edges (black squares). A specific path of nodes and edges is highlighted in red. This path is mapped to a sequence of matrices: a 4x2 matrix, a 4x2 matrix, a 4x2 matrix, and a 4x2 matrix. Arrows show how the red path in the graph corresponds to the red entries in the matrices. The matrices are labeled 'matr' and 'cognome'.

matr		matr	cognome	...
$-\infty$		003	Neroni	...
031		005	Verdi	...
100		009	Carini	...
		010	Rossi	...

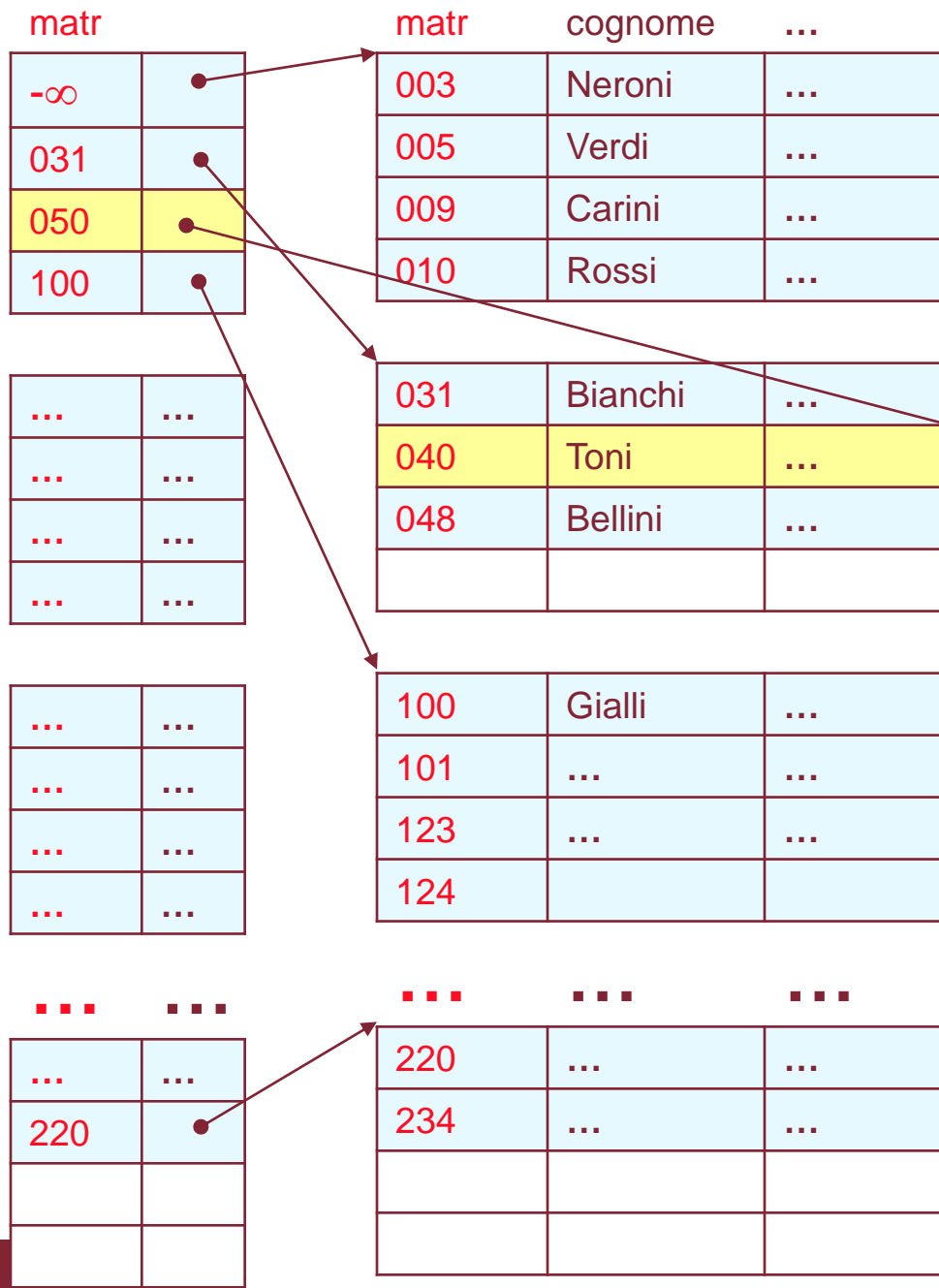
...	...
...	...
...	...
...	...

...	...
...	...
...	...
...	...

...	...
...	...
...	...
...	...

...	...
220	

...
220
234



Inserimento

... occorre richiedere un nuovo blocco al file system, ripartire i record tra vecchio e nuovo blocco e riscrivere tutti i blocchi modificati.

matr		matr	cognome	...
-∞	●	003	Neroni	...
031	●	005	Verdi	...
050	●	009	Carini	...
100	●	010	Rossi	...

...	...
...	...
...	...
...	...

031	Bianchi	...
040	Toni	...
048	Bellini	...

...	...
...	...
...	...
...	...

100	Gialli	...
101
123
124		

...	...
220	●

220
234

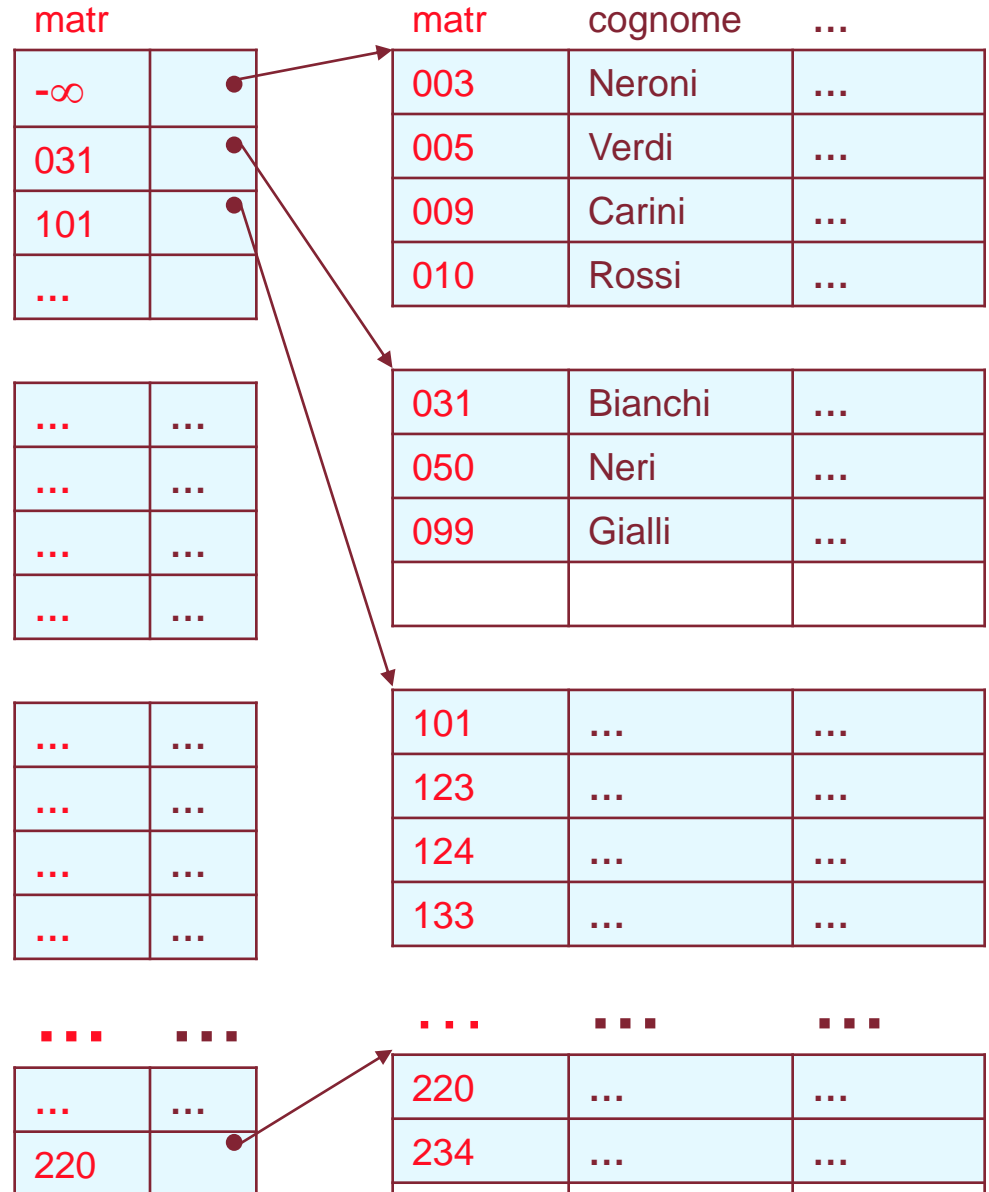
050	Neri	...
099	...	

... occorre richiedere un nuovo blocco al file system, ripartire i record tra vecchio e nuovo blocco e riscrivere tutti i blocchi modificati.

Cancellazione

050	Neri	
-----	------	--

costo per la ricerca ...



Cancellazione

...+1

(per scrivere
il blocco modificato)

matr		cognome	...
-∞		003	Neroni
031		005	Verdi
101		009	Carini
...		010	Rossi

...	...
...	...
...	...
...	...

...	...
...	...
...	...
...	...

...	...
220	

031	Bianchi	...
099	Gialli	...

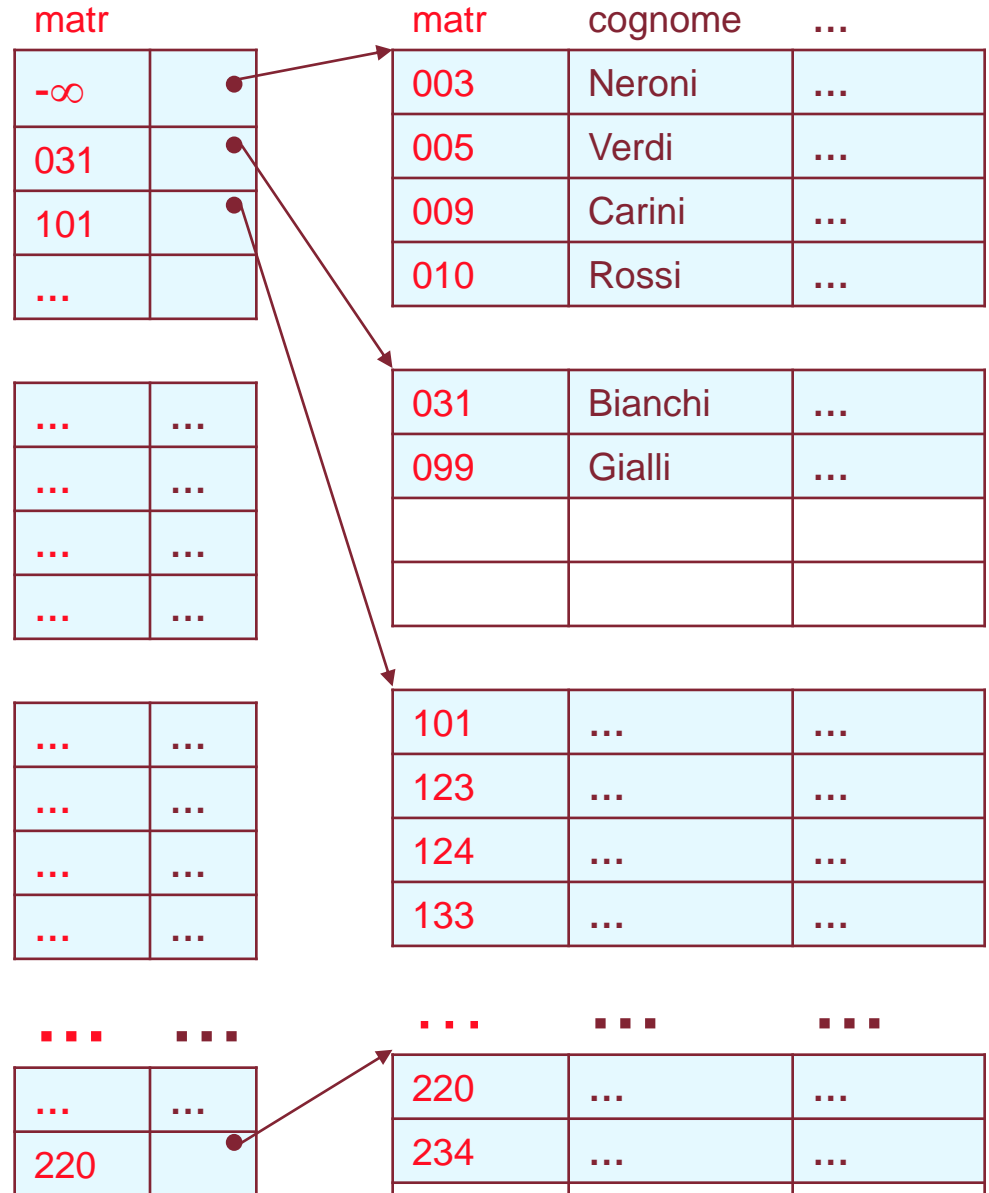
101
123
124
133

...
220
234

Cancellazione

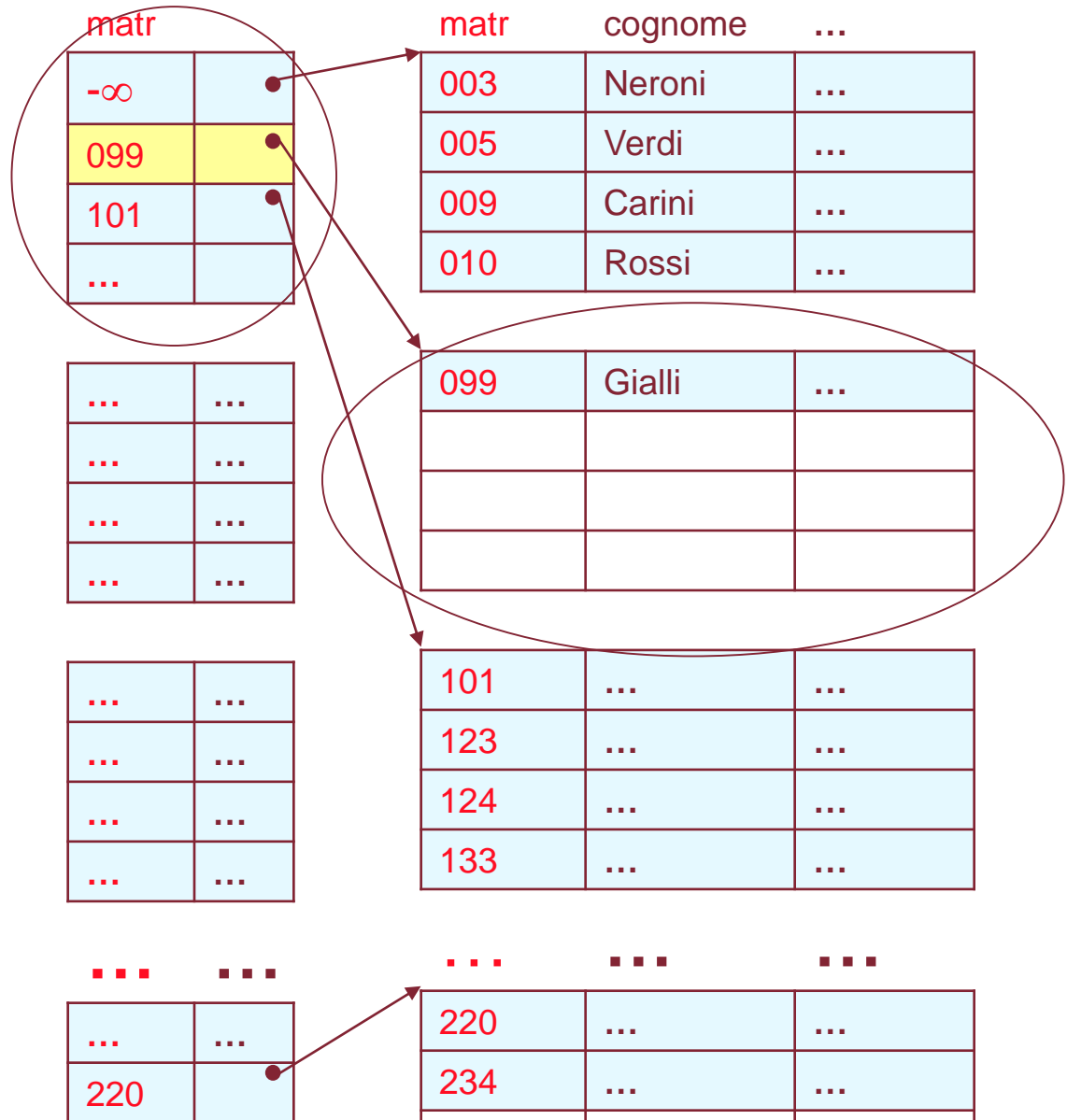
031	Bianchi	...
-----	---------	-----

Se il record
cancellato è il primo
di un blocco ...



Cancellazione

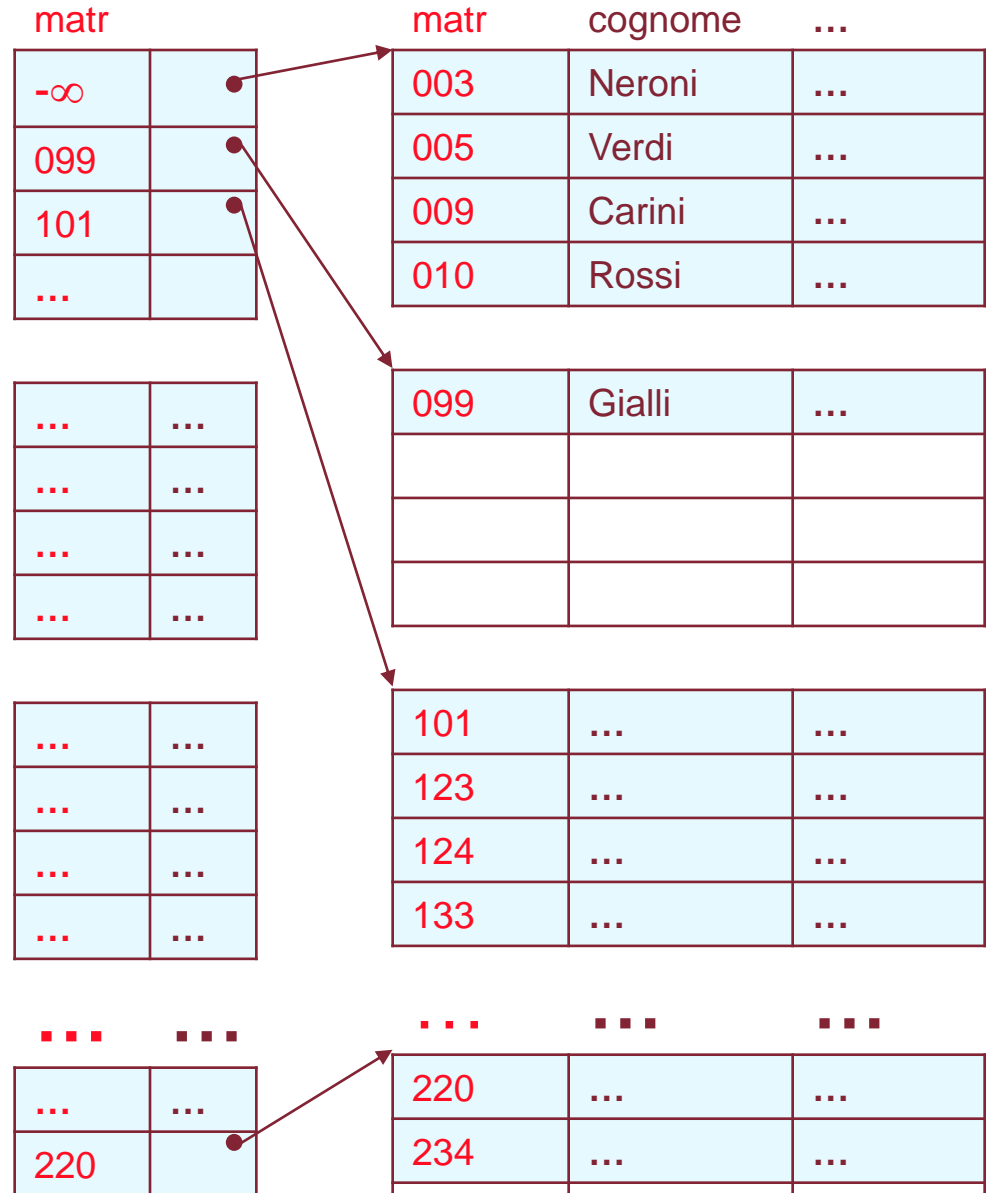
... sono necessari
ulteriori accessi



Cancellazione

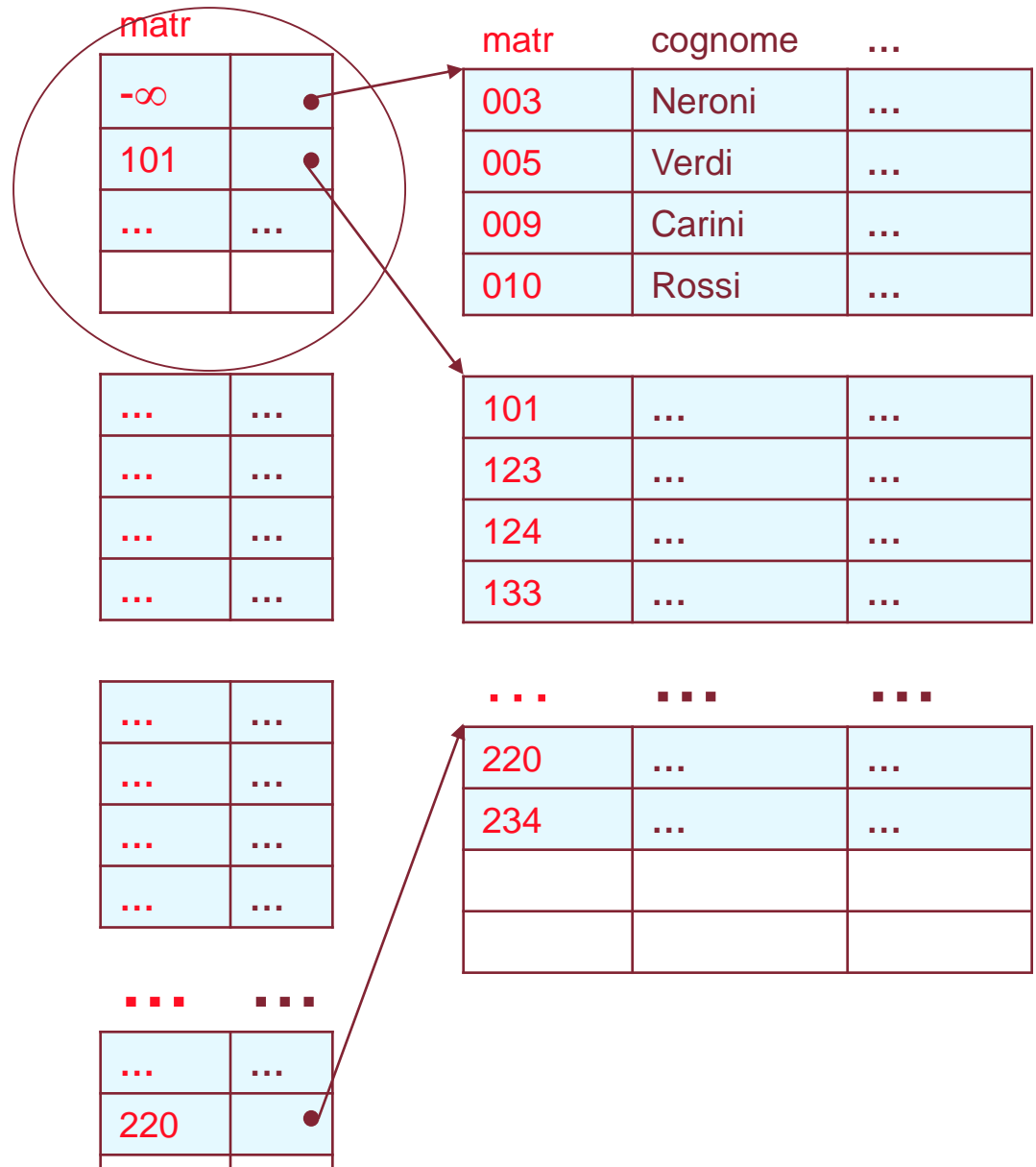
099	Gialli	...
-----	--------	-----

Se il record cancellato
è l'unico record
del blocco ...

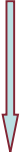


Cancellazione

... il blocco viene
restituito al sistema e
viene modificato
il file indice



Modifica (non coinvolge la chiave)

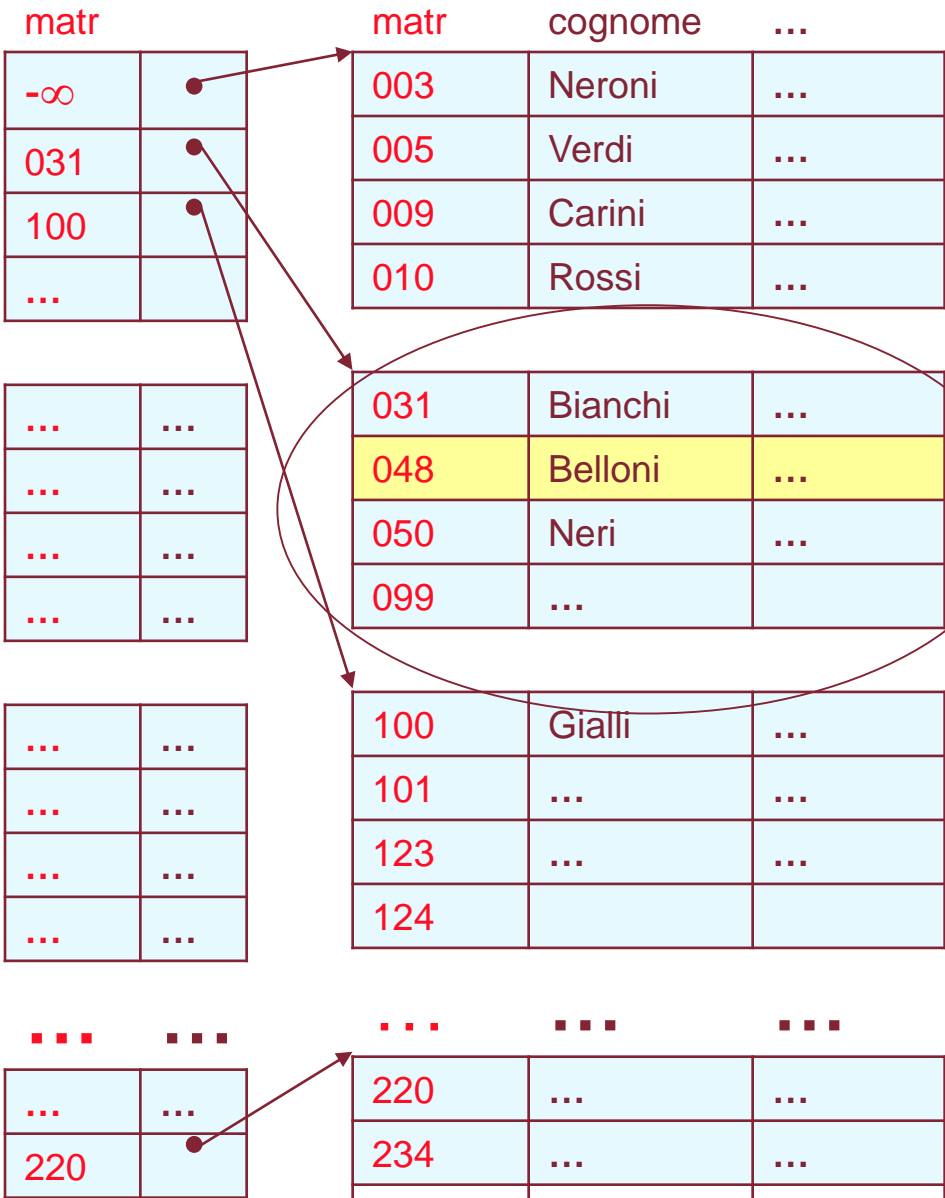
048	Bellini	
		
048	Belloni	

costo per la ricerca ...



Modifica (non coinvolge la chiave)

...+1 per riscrivere il blocco modificato



Se la modifica coinvolge la **chiave**, allora

cancellazione + inserimento

- Consideriamo ora il caso in cui il file principale contiene record **puntati**.
- Nella fase di inizializzazione è preferibile lasciare **più spazio libero** nei blocchi per successivi inserimenti.
- Poiché i record sono puntati, **non possono essere spostati** per mantenere l'ordinamento quando si inseriscono nuovi record
- Se **non c'è spazio sufficiente** in un blocco B per l'inserimento di un **nuovo** record, occorre richiedere al sistema un **nuovo blocco** che viene **collegato** a B tramite un **puntatore**; in tal modo **ogni record del file indice punta al primo blocco di un bucket** e il file indice **non viene mai modificato** (a meno che le dimensioni dei bucket non siano diventate tali da richiedere una riorganizzazione dell'intero file).

- La **ricerca** di un record con chiave v richiede la **ricerca sul file indice** di un **valore della chiave che ricopre v** e quindi la **scansione del bucket corrispondente**.
- La **cancellazione** di un record richiede la **ricerca** del record e quindi la **modifica dei bit di cancellazione nell'intestazione del blocco**.
- La **modifica** di un record richiede la **ricerca** del record; quindi, se la modifica **non coinvolge** campi della chiave, il record viene modificato e il blocco riscritto. **Altrimenti** la modifica equivale ad una **cancellazione seguita da un inserimento**. In quest'ultimo caso **non è sufficiente** modificare il bit di cancellazione del record cancellato, ma **è necessario** inserire in esso un **puntatore al nuovo record** inserito in modo che questo sia raggiungibile **da qualsiasi record che contenga un puntatore al record cancellato**.
- Poiché **non è possibile mantenere il file principale ordinato**, se si vuole avere la possibilità di esaminare il file **seguendo l'ordinamento della chiave** occorre inserire **in ogni record un puntatore al record successivo** nell'ordinamento.

Indice

matr

File principale

matr

cognome

...

blocco 1

-∞	•
031	•
101	•
...	

003	Neroni	...
005	Verdi	...

009	Carini	...
010	Rossi	...

blocco 2

...	...
...	...
...	...
...	...

031	Bianchi	...
048	Bellini	...
050
099

blocco 3

...	...
...	...
...	...
...	...

101
123
124
133

...

blocco m

...	...
220	•

220
234



- Quando i record del file principale sono puntati, una volta inseriti non possono essere spostati per mantenere l'ordinamento (quindi in pratica l'ordinamento stretto dei record vale solo all'inizializzazione).
- Viceversa, a **differenza** da quanto accade per l'ISAM classico, i record dei blocchi **indice non vengono mai modificati**, quindi ciò che **rimane valido** è la **ripartizione degli intervalli delle chiavi**.
- Se un record indice punta ad un'area di dati con valori di chiave comprese tra k_1 e k_2 , questa condizione **deve rimanere valida**.
- Se il blocco originario si riempie, e arrivano nuovi record con valori di chiave compresi sempre tra k_1 e k_2 , dobbiamo **allocare un nuovo blocco** che però, **anzichè** essere puntato **dall'indice**, sarà **linkato al blocco originario**, e così per ogni nuovo blocco che si riempie (abbiamo cioè una **lista di blocchi di overflow** che partono da quello originario, dove ognuno punta al successivo).

Esempio 1



Supponiamo di avere un file di 150.000 record. Ogni record occupa 250 byte, di cui 50 per il campo chiave. Ogni blocco contiene 1024 byte. Un puntatore a blocco occupa 4 byte.

- a) Se usiamo un indice ISAM sparso, e assumiamo che i record non siano puntati e che il fattore di utilizzo dei blocchi del file principale sia 0,7 (cioè i blocchi non sono completamente pieni, **ma pieni al più** al 70%), quanti blocchi dobbiamo utilizzare per **l'indice** ?
- b) Se usiamo un indice ISAM sparso, e assumiamo che i record siano **puntati** e che i blocchi del file principale siano **pieni**, quanti blocchi dobbiamo utilizzare per **l'indice** ?
- c) Se utilizziamo la ricerca binaria, quale è il numero **massimo** di accessi a blocco per ricercare un record presente nel file nei casi a) e b), supponendo nel caso b) di non avere liste di overflow ?

Esempio 1



Abbiamo i seguenti dati:

- il file contiene 150.000 record: $NR = 150.000$
- ogni record occupa 250 byte: $R = 250$
- il campo chiave occupa 50 byte: $K = 50$
- ogni blocco contiene 1024 byte: $CB = 1024$
- un puntatore a blocco occupa 4 byte: $P = 4$

- a) I record sono di taglia **fissa**, quindi non occorrono puntatori all'inizio del blocco; inoltre, **non essendo stato altrimenti specificato** nella traccia, **assumiamo che un record non possa superare i limiti di un blocco** (quindi ogni blocco contiene **un numero intero di record**). Poiché i record **non sono puntati**, possono essere spostati, e quando un blocco si riempie, ne allochiamo **uno nuovo** che comporterà un aggiornamento dell'indice. Di conseguenza, nei blocchi del file principale **non occorre** un puntatore **al prossimo blocco**. L'esercizio fornisce il fattore di occupazione attuale dei blocchi del file del file principale, ma non indica un fattore di occupazione per i blocchi **indice**, quindi possiamo assumere che siano **pieni**.

a) Cont.

- Dobbiamo stabilire quanti **record indice** occorrono, quindi sapendo che l'indice contiene un **record per ogni blocco del file principale**, dobbiamo calcolare quanti sono questi blocchi. Sappiamo che i blocchi dati sono occupati **al più** al 70% (indichiamo questa quantità con PO), quindi prima di tutto vediamo quanti record **interi** possono essere contenuti al **massimo** nella porzione indicata di blocco. Indicando con M questo numero, avremo che $M = (CB \times PO)/R$, quindi nel nostro caso $M = \lfloor (1024 \times 70/100) \rfloor / 250 = (716/250) = 2,86$. Prendiamo la parte intera **inferiore** perché la percentuale indica lo spazio **massimo** occupato
- Poiché M deve essere **intero**, avremo $M = 2$. Anche in questo caso abbiamo preso la parte **intera inferiore** del risultato per essere sicuri di non superare l'occupazione massima richiesta

a) Cont.

- Il numero di blocchi da indicizzare, indicato con BF, sarà quindi $BF = \lceil NR/M \rceil = \lceil 150.000/2 \rceil = 75.000$. Abbiamo bisogno quindi di **75.000 record indice**. Abbiamo preso la parte intera **superiore** perché assumiamo che i blocchi vengano allocati per intero.
- Ogni record **indice** è composto da chiave e puntatore al blocco, quindi occupa $RI = K + P$ byte, cioè **RI = 54 byte**. Dobbiamo ora calcolare **quanti record indice interi** possono essere contenuti in un **blocco pieno** (se anche in questo caso fosse precisata una **percentuale di utilizzo**, il calcolo **procederebbe in maniera analoga** a quella usata per i blocchi dati). Indicando con MI questo numero, avremo $MI = \lfloor CB/RI \rfloor = \lfloor 1024/54 \rfloor = 18$ **record indice per blocco**. In definitiva, indicando con BI il numero di blocchi indice, avremo $BI = \lceil BF/MI \rceil = \lceil 75.000/18 \rceil = 4167$ blocchi indice;



- **Nota:** il calcolo basato sul numero complessivo di byte necessari per l'indice $RI \times NR$ diviso per la capacità CB di ogni blocco darebbe un numero di blocchi pari a $\lceil (54 \times 75.000) / 1024 \rceil = 3956$, ma presenterebbe un errore di fondo, in quanto non assicurerebbe che ogni record sia contenuto interamente in un blocco;

- b) ogni blocco del file principale deve essere visto come **parte di un bucket**, e quindi dobbiamo anche prevedere **spazio per un puntatore**. L'esercizio ci dice in questo caso che i blocchi dati sono pieni, e lo stesso possiamo assumere per i blocchi indice, visto che non è specificato altrimenti.
- Altra assunzione che possiamo fare, visto che **non è stato specificato altrimenti**, è che **non ci siano** liste di overflow, cioè che **tutti i record di dati** siano contenuti in blocchi puntati direttamente dall'indice.
- Vediamo prima di tutto in queste nuove condizioni **quanti record interi di dati entrano al massimo in un blocco**.

- b) cont.
 - ogni blocco del file principale deve essere visto come **parte di un bucket**, e quindi dobbiamo anche prevedere **spazio per un puntatore**. L'esercizio ci dice in questo caso che i blocchi dati sono pieni, e lo stesso possiamo assumere per i blocchi indice, visto che non è specificato altrimenti.
 - Altra assunzione che possiamo fare, visto che **non è stato specificato altrimenti**, è che **non ci siano** liste di overflow, cioè che **tutti i record di dati** siano contenuti in blocchi puntati direttamente dall'indice.
 - Vediamo prima di tutto in queste nuove condizioni **quanti record interi di dati entrano al massimo in un blocco**.

- b) cont.
 - $M = \lfloor (CB-P)/R \rfloor = \lfloor 1020/250 \rfloor = \lfloor 4,08 \rfloor = 4$
 - Vediamo allora quanti blocchi vanno indicizzati (nel caso di non avere liste di overflow). Indicando questo numero con BF avremo $BF = \lceil NR/MP \rceil = \lceil 150.000/4 \rceil = 37.500$.
 - A questo punto calcoliamo il numero di blocchi indice. **La struttura di blocco indice è identica** a quella del caso precedente, quindi ricordiamo che la taglia del record indice è $RI = 54$ byte , che il numero massimo di record indice per blocco è $MI = 18$ e quindi occorrono $BI = \lceil BF/MI \rceil = \lceil 37.500/18 \rceil = 2084$ blocchi indice;

Esempio 1



- c) in entrambi i casi la ricerca si effettua prima di tutto sui blocchi **indice**; assumiamo una **ricerca binaria**; i due casi si differenziano, perché nel caso di record **non puntati** dobbiamo ancora leggere in memoria **un solo blocco di record di dati**, mentre **nel caso di record puntati** l'indice punta ad un bucket che potrebbe contenere **più blocchi di overflow**, che vanno tutti esaminati.
 - Nel nostro caso però l'esercizio dice esplicitamente che non ci sono liste di overflow, quindi in entrambe le configurazioni dobbiamo aggiungere un solo accesso a blocco, quindi nella configurazione a) avremo $A = \lceil \log_2 BI \rceil + 1 = \lceil \log_2 4167 \rceil + 1 = 14$, mentre nella configurazione b) avremo $A = \lceil \log_2 BI \rceil + 1 = \lceil \log_2 2084 \rceil + 1 = 13$.

Esempio 2



Supponiamo di avere un file di 200.000 record. Ogni record occupa 150 byte, di cui 40 per il campo chiave. Ogni blocco contiene 512 byte. Un puntatore a blocco occupa 4 byte. Usiamo un indice ISAM sparso, e assumiamo che i record non siano puntati e che il fattore di utilizzo sia dei blocchi del file dati sia dei blocchi dell'indice sia 0,8 (cioè i blocchi non sono completamente pieni, ma pieni al più all' 80%).

- a) Quanti blocchi dobbiamo utilizzare per il file dati (file principale)?
- b) Quanti blocchi dobbiamo usare per l'indice ?
- c) Calcolare il numero massimo di accessi per la ricerca di un record nel file principale, utilizzando la ricerca binaria sul file indice.

Esempio 2



Abbiamo i seguenti dati:

Numero record	NR = 200.000
Taglia record	R = 150 byte
Taglia chiave	K = 40 byte
Taglia puntatore	P = 4 byte
Capacità blocco	CB = 512 byte.

L'esercizio dice che i blocchi sono pieni **al più** all'80%, quindi la capacità che dobbiamo considerare è 409,6, approssimata per **difetto**, quindi teniamo conto di una capacità $CB = 409$.

Capacità blocco	CB = 409 byte.
-----------------	----------------

Esempio 2



a) Prima di tutto dobbiamo calcolare quanti record (RB) entrano in un blocco con la nuova capacità calcolata, quindi $RB = \lfloor CB/R \rfloor = \lfloor 409/150 \rfloor = 2$ (prendiamo la parte intera inferiore perché i record devono essere interi). A questo punto servirà un numero BF di blocchi per il file principale dato da

$BF = \lceil NR/ RB \rceil = \lceil 200.000/2 \rceil = 100.000$ (prendiamo la parte intera superiore perché non possiamo allocare porzioni di blocco).

b) Per **ogni blocco del file principale** abbiamo bisogno di un **record indice nel file indice**. Innanzitutto dobbiamo calcolare la **taglia dei record indice (RI)**. Il record indice è composto da **una chiave e da un puntatore**, quindi $RI = K + P = 44$ byte. A questo punto calcoliamo quanti record indice (IB) entrano in un blocco

$$IB = \lfloor CB / RI \rfloor = \lfloor 409 / 44 \rfloor = 9$$

I blocchi necessari per l'indice (BI) saranno allora $BI = \lceil BF / IB \rceil = \lceil 100.000 / 9 \rceil = 11.112$

Esempio 2



b) Per **ogni blocco del file principale** abbiamo bisogno di un **record indice nel file indice**. Innanzitutto dobbiamo calcolare la **taglia dei record indice (RI)**. Il record indice è composto da **una chiave e da un puntatore**, quindi $RI = K + P = 44$ byte. A questo punto calcoliamo quanti record indice (IB) entrano in un blocco $IB = \lfloor CB / RI \rfloor = \lfloor 409 / 44 \rfloor = 9$

- I blocchi necessari per l'indice (BI) saranno allora $BI = \lceil BF / IB \rceil = \lceil 100.000 / 9 \rceil = 11.112$

c) Il numero massimo di accessi (MA) utilizzando la ricerca binaria è dato dagli accessi necessari per la ricerca nell'indice più un accesso al file principale, quindi $MA = \lceil \log_2 BI \rceil + 1 = \lceil \log_2 11.112 \rceil + 1 = 14 + 1 = 15$

Il calcolo del logaritmo in base 2 è banale anche senza calcolatrice (si richiede la parte intera superiore, non il valore esatto!) quindi è sempre richiesto negli esercizi di esame

$2^0=1$, $2^1=2$, $2^2=4$, $2^3=8$, $2^4=16$... mi fermo quando ho superato il numero di cui devo calcolare la parte intera superiore del logaritmo in base 2