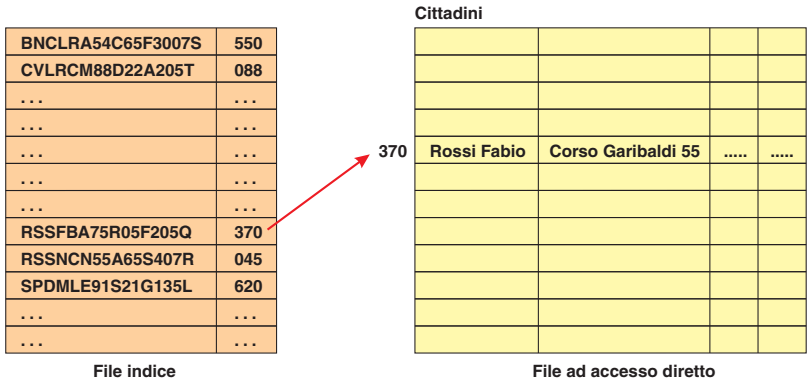
Organizzazione dei file per l'accesso a chiave

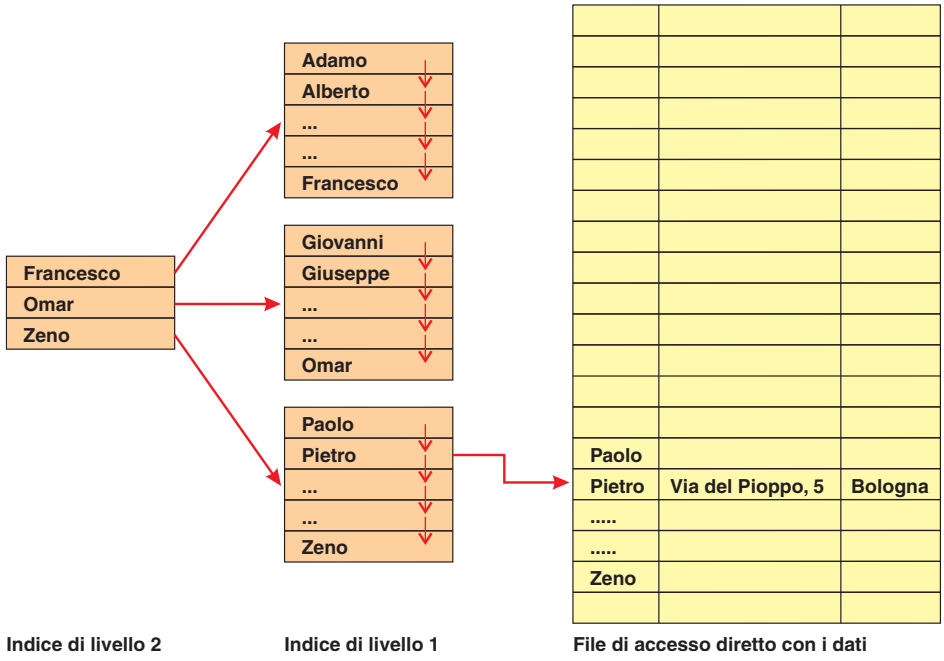
L'accesso a chiave è di importanza fondamentale per le applicazioni. Per esempio per accedere alle informazioni anagrafiche di un cittadino, l'impiegato dell'Ufficio anagrafe recupera le informazioni desiderate in base al valore del codice fiscale. Quindi, per recuperare nel file anagrafico di nome *Cittadini* il record del signor *Rossi Fabio*, che ha codice fiscale RSSFBA75R05F205Q, l'applicazione che visualizza le informazioni esegue il comando:

```
read  ('Cittadini','RSSFBA75R05F205Q',  buffer);
```

Il metodo di accesso determina la posizione del record cercato in base al valore del campo chiave. Questa funzionalità può essere implementata mediante un'opportuna organizzazione del file che prende il nome di organizzazione **sequenziale a indice**. Un *indice* consiste, concettualmente, in una struttura dati aggiuntiva rispetto al file che contiene i dati e che serve al sistema per riuscire a risalire rapidamente alla posizione del record cercato.

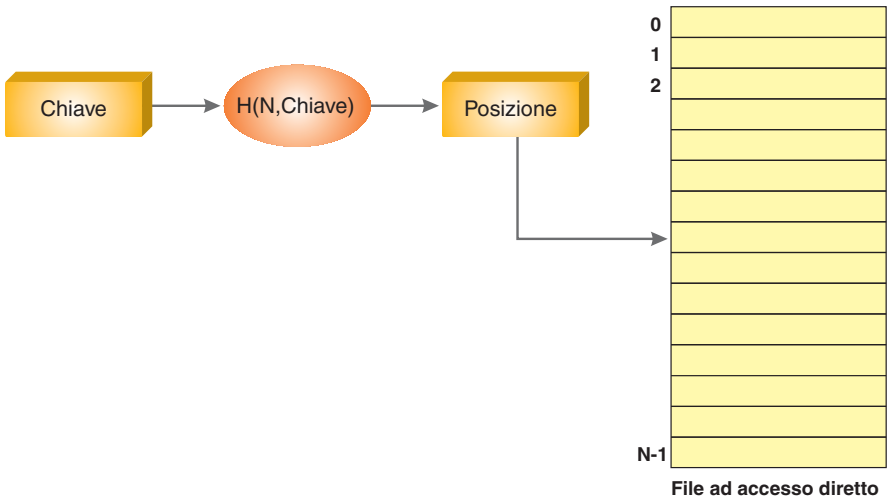


Il *file indice* memorizza le chiavi di accesso al file *Cittadini*. Il sistema inserisce i record anagrafici in un file ad accesso diretto e memorizza la corrispondente chiave nel *file indice* mantenendolo ordinato per valori crescenti di chiave. Ogni riga del file indice contiene, accanto alla chiave un puntatore che indica la posizione occupata dal record con i dati nel file ad accesso diretto. Per trovare il record del signor *Rossi* l'algoritmo di ricerca effettua una *ricerca binaria* nel file indice e rileva che i dati cercati sono memorizzati nel record 370 del file *Cittadini*. Il problema dell'accesso a chiave è così ricondotto a quello all'accesso al record *n*-esimo in un file ad accesso diretto.
La ricerca binaria in un file con N record prevede di controllare il valore cercato nel record di posto N/2. Se il dato cercato è memorizzato in quella posizione la ricerca è finita. Diversamente, essendo i dati ordinati, la chiave cercata si trova in un insieme formato da N/2 record. L'algoritmo di ricerca itera il procedimento dimezzando ogni volta l'intervallo di ricerca. Ragionando in questo modo si dimostra che la ricerca binaria permette di identificare il valore cercato in un elenco di N chiavi in non più di $log_2 N$ passi. Questo significa, in pratica, che per trovare una chiave in un elenco di circa 1.000.000 di chiavi servono al massimo 20 passi. In pratica la tecnica usata è lievemente diversa: quando l'intervallo di ricerca è ridotto a un elenco di poche decine di valori, la ricerca binaria è interrotta e il valore è cercato con una scansione lineare dell'elenco.
Al crescere della dimensione del file l'organizzazione sequenziale ad indice può diventare inefficiente: in lettura, a causa del crescere del numero di accessi al disco necessari per identificare la chiave nel file indice, e in scrittura, per la necessità di mantenere ordinato un file di dimensioni crescenti. L'organizzazione ad **indici su più livelli** cerca di risolvere questi problemi riducendo la dimensione dello spazio di ricerca per gli indici. L'idea è, in linea di principio, molto semplice: il file degli indici è a sua volta indicizzato costruendo un indice (indice di livello 2) per accedere al file degli indici vero e proprio (indice di livello 1). Se anche l'indice di livello 2 è troppo grande si può costruire un indice di livello 3 per accedere all'indice di livello 2 che punta all'indice vero e proprio.



Per accedere al record che ha come codice *Pietro* si scandisce l'indice di livello 2 alla ricerca di un valore maggior od uguale a *Pietro*; *Zeno* soddisfa a questa condizione. Il puntatore accanto a *Zeno* indica in quale blocco dell'indice di livello 1 si deve proseguire la ricerca. La ricerca continua nel blocco dove sono conservati gli indici veri e propri e tra questi si trova quello che permette di accedere al record con i dati di *Pietro*.
Si osservi anche la catena di collegamenti tra gli indici di livello 1 che permette la lettura sequenziale del file ordinato per valore crescente di chiave, a partire da qualsiasi valore della chiave.

Un altro modo di usare i file ad accesso diretto per implementare l'accesso a chiave consiste nell'usare una funzione che trasforma la chiave in un valore numerico e di usare tale valore come posizione occupata dal record memorizzato in un file ad accesso diretto. Questa tecnica di indicizzazione si chiama *metodo di trasformazione della chiave* o metodo **hash:** una funzione di *hashing* , avente come argomento una chiave, deve restituire la posizione del record corrispondente, con l'obiettivo di distribuire nel modo più uniforme possibile i record all'interno del file.

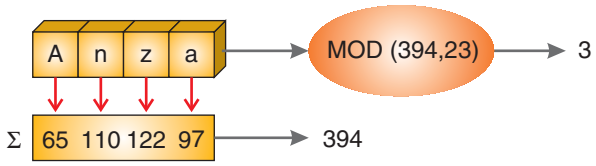


I record sono memorizzati in un file ad accesso diretto capace di memorizzare N record. La funzione di hashing *H(N,Chiave)* calcola un numero di record sulla base del valore della chiave e del massimo numero di record previsti per il file, cioè trasforma l'intervallo dei valori della chiave in un numero compreso tra 0 e N-1, dove N è il numero di record nel file.

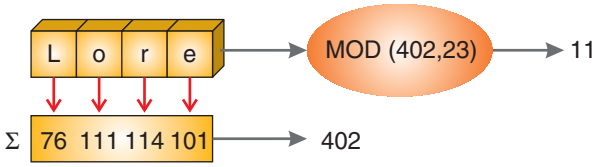
Poiché il calcolo dipende dal numero massimo di record nel file, quando si crea un file con questa organizzazione occorre specificare qual è la dimensione massima del file.

Una semplice funzione di hashing utilizza un numero intero ricavato dalla chiave, divide questo intero per il numero massimo di record e usa il resto della divisione come valore restituito dalla funzione. Nel caso di chiavi alfanumeriche si costruiscono funzioni di hashing basate sul valore dei codici ASCII (o di un'altra codifica) dei caratteri che compongono la chiave.

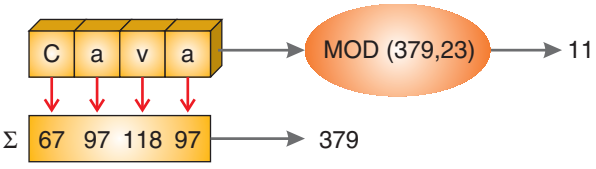
Supponiamo, per esempio, che la chiave sia composta da 4 caratteri alfabetici; si divide la somma dei codici ASCII dei caratteri componenti la chiave per il numero di record previsti, ottenendo come resto della divisione un valore intero H che fornisce la posizione del record nel file. Applichiamo questa tecnica ai 15 record del file *Anagrafe* usato nell'esempio precedente per l'accesso a chiave. Cominciamo a calcolare la posizione occupata dal record di chiave *Anza*. Per ragioni che saranno chiarite in seguito vogliamo inserire i record in un hash file con 23 record.



La somma dei codici ASCII dei caratteri che compongono la chiave vale 394 e il resto della divisione tra 394 e 23 vale 3. Il record con i dati di *Anzani* è memorizzato nel record 3 del file. Applicando il medesimo algoritmo alla chiave di *Lorenzetti* 'Lore' si osserva che la posizione occupata dal record nel file *Anagrafe* è data dal resto della divisione tra 402 e 23 cioè 11.



Quando un record viene aggiunto in un file gestito con una tecnica hashing, il software di gestione dell'accesso tenta di inserire il record nella posizione calcolata dalla funzione di hashing. Se il posto è già occupato da un altro record, accade una **collisione** e il record deve essere messo in un'altra posizione del file. Calcoliamo, per esempio, la posizione di inserimento del record con chiave "Cava".



In questo caso si ha una collisione con il record di chiave "Lore". Estendiamo il procedimento a tutti i record del file. I risultati dell'esercizio sono riportati nella tabella sottostante che mostra, accanto alle posizioni calcolate con l'algoritmo illustrato (colonna Σ mod 23), anche il caso di posizioni calcolate con il medesimo algoritmo ma usando un file più piccolo, composto solamente da 17 record (colonna Σ mod 17). Le collisioni sono evidenziate in rosso.

Se i 15 record del file sono collocati in un file di 17 record avvengono 5 collisioni. Le collisioni scendono però a due se si usa un file con 23 record. Il risultato intuitivo ha carattere di generalità ed è confermato dalle raccomandazioni dei manuali che suggeriscono, nella creazione di un file indicizzato hashing, di incrementare il numero dei record previsti del 30% e di arrotondare in eccesso il valore ottenuto ad un numero primo. Per esempio con un file di 15 record:

$$15 \times 1{,}3 = 19{,}5 \rightarrow 23$$

si ottiene 23 come dimensione raccomandata per il file. Il suggerimento, che ha lo scopo di ridurre il numero di possibili collisioni, è frutto di considerazioni di tipo statistico sulle prestazioni delle funzioni di hashing e, conseguentemente, non esclude che per particolari distribuzioni di valori delle chiavi si manifestino in ogni caso molte collisioni. Ci sono diverse tecniche per la **gestione delle collisioni.** La più diffusa nelle implementazioni di indicizzazione con funzione di hashing, consiste nel creare un'area separata, detta **area di overflow**, usata per contenere i record che hanno valori di chiave che collidono con le chiavi già esistenti. Applicando questa tecnica ai dati dell'esempio si arriva al file in figura.

Codice	Σ mod 23	Σ mod 17
Anza	3	3
Berg	16	10
Bian	10	4
Carr	1	1
Catt	5	5
Cava	11	**5**
Dott	20	**3**
Ermo	12	12
Gana	7	**1**
Lore	**11**	11
Magn	19	13
Mare	21	15
Ross	9	**15**
Ros2	13	**1**
Vene	**7**	7

Gana → 7
Vene → 7
Cava → 11
Lore → 11

	Codice	Cognome	Nome	...	Overflow
0					=
1	Carr	Carrara	Alessandro		=
2					=
3	Anza	Anzani	Antonio		=
4					=
5	Catt	Cattaneo	Mirella		=
6					=
7	**Gana**	Canapini	Walter		**24**
8					=
9	Ross	Rossi	Giuliano		=
10	Bian	Bianchi	Francesca		=
11	**Cava**	Cavallotti	Ennio		**23**
12	Ermo	Ermolli	Marco		=
13	Ros2	Rossi	Piercarlo		=
14					=
15					=
16	Berg	Bergantini	Mario		=
17					=
18					=
19	Magn	Magnani	Gianni		=
20	Dott	Dotti	Laura		=
21	Mare	Marenzi	Giuliana		=
22					=
23	**Lore**	Lorenzetti	Carla		=
24	**Vene**	Venezian	Luca		=
25					=
26					=

Area di overflow