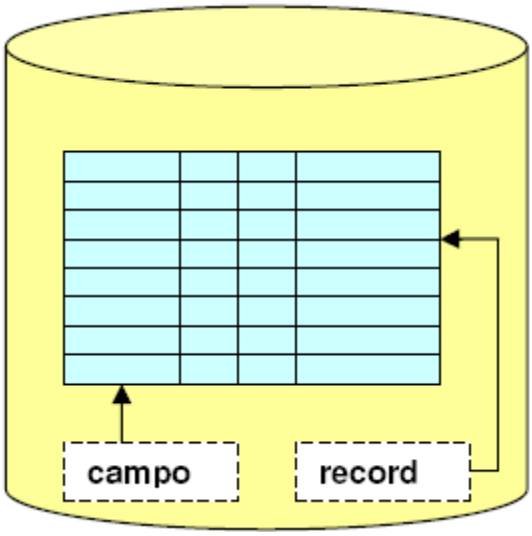
ORGANIZZAZIONE E GESTIONE DEGLI ARCHIVI

Un **archivio** di dati è un file (memorizzato su un supporto fisico) che contiene informazioni su una determinata entità. Un archivio è costituito da un insieme di registrazioni o **record** (record logici) ciascuno dei quali è costituito da un insieme predefinito e strutturato di informazioni elementari dette attributi o **campi**. Ogni campo contiene un'informazione elementare.
Un record fisico è l'insieme dei dati letti o scritti ad ogni lettura o scrittura fisica sul supporto (un record fisico corrisponde ad un **blocco**, incrocio traccia-settore, ed ha dimensione fissa, generalmente 512, 1024 o 2048 byte). Un record fisico può contenere più record logici, può coincidere con il record logico, può essere parte di un record logico.



Le operazioni possibili su un archivio sono:

- **creazione dell'archivio**
- **registrazione di un record**
- **lettura di un record**
- **aggiornamento dei campi di un record**
- **cancellazione di un record**
- **visita di tutti i record dell'archivio**.

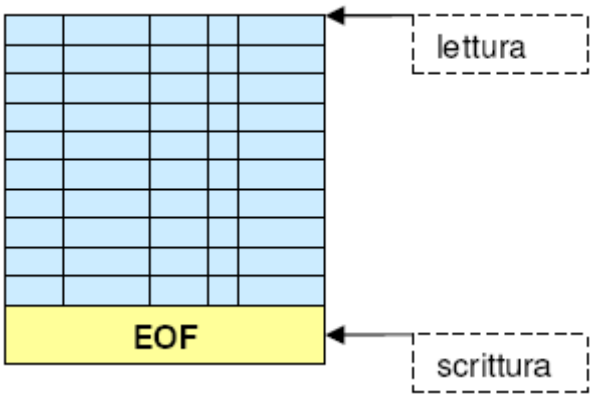
L'archivio, prima dell'utilizzazione, deve essere "*aperto*" con una opportuna istruzione di apertura e dopo l'utilizzazione deve essere "*chiuso*".

Esistono diverse **organizzazioni degli archivi:**

- **sequenziale**
- **casuale**
- **a indici**
- **sequenziale con indice -**

Organizzazione SEQUENZIALE (1)

In un archivio **sequenziale** tutti i record sono memorizzati e reperiti sequenzialmente, uno di seguito all'altro. In memorizzazione i record vengono aggiunti in fondo all'archivio,dopo l'ultimo.
In lettura si parte sempre dal primo record e, dopo ogni lettura, si passa automaticamente al successivo; questo significa che se si vuol leggere l'n-esimo record occorre sempre leggere anche tutti i record che lo precedono. L'archivio termina con **EOF** (*End Of File*), una speciale marca di fine file.

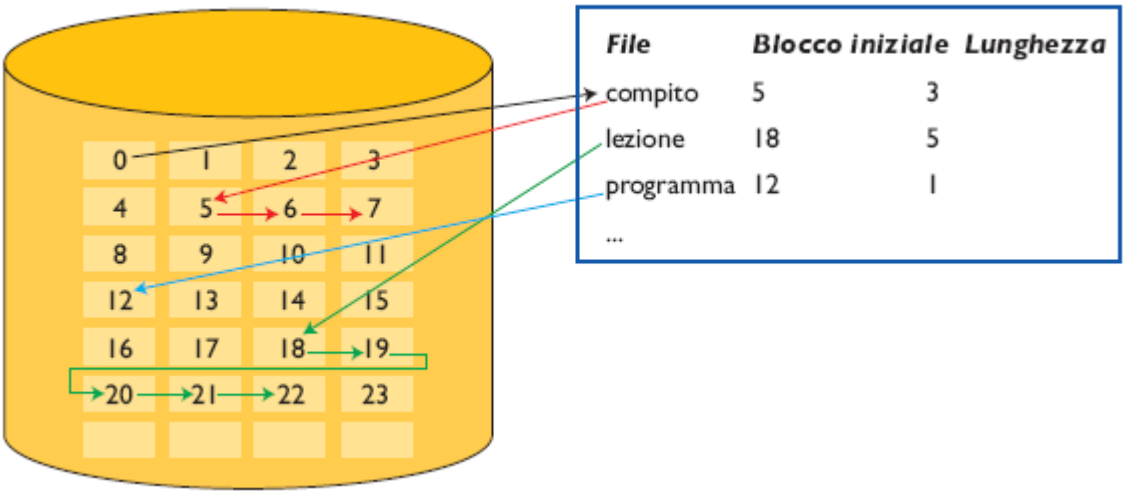


L'archivio può essere aperto in *input/read* (per lettura, partendo dall'inizio), in *output/write* (per scrittura partendo dall'inizio: se l'archivio non esiste lo crea, altrimenti lo cancella), in *append* (per scrittura partendo dalla fine).

Le istruzioni sono: *lettura record* corrente (e dopo ci si posiziona automaticamente al successivo) e *scrittura record* (che viene aggiunto in fondo). La maggior parte dei linguaggi di programmazione non prevede la riscrittura e la cancellazione dei record; alcuni, quali il Cobol, sì.
La lettura dei record termina quando si raggiunge l'EOF (che deve essere controllato prima di leggere il record, per vedere se si è già arrivati alla fine).

L'organizzazione fisica sequenziale

Occupiamoci, ora, dell'aspetto fisico dell'archivio.
Con **archivio fisico sequenziale** si indica il modo in cui i file sono scritti su disco e non la natura logica degli stessi, descritta nel precedente paragrafo di questa lezione. In questo paragrafo, pertanto, quando parliamo di "archivi sequenziali" facciamo riferimento ad "archivi fisici sequenziali".
Gli archivi sequenziali sono memorizzati sulla memoria di massa attraverso la **tecnica dell'allocazione contigua**. In questo modo, i blocchi che costituiscono il file sono memorizzati uno di seguito all'altro, ossia in ordine sequenziale, come si osserva nella figura seguente.



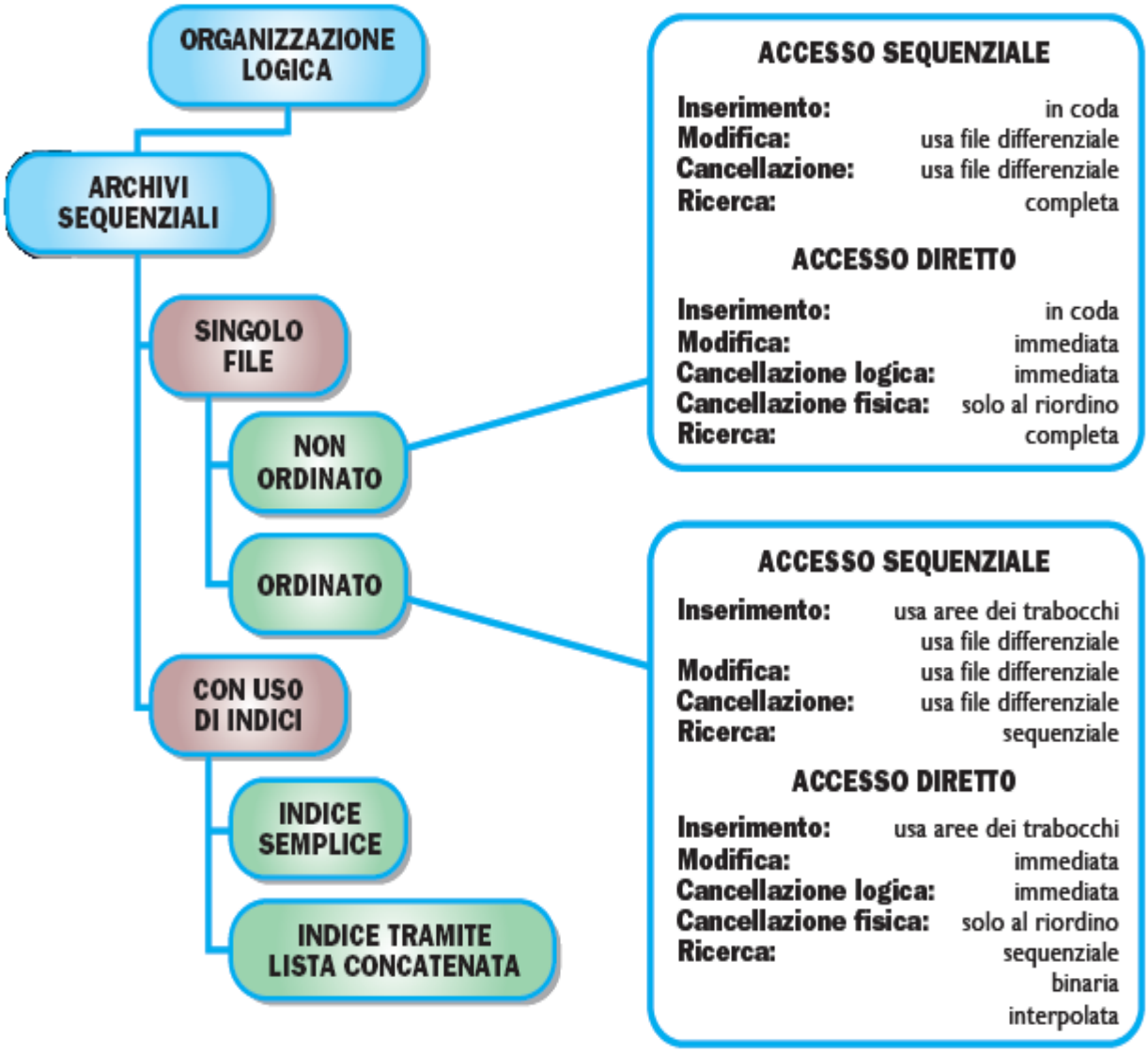
L'accesso a un file allocato in modo contiguo è, quindi, piuttosto semplice, ma le difficoltà sorgono ogniqualvolta si ha la necessità di allocare un nuovo file. Se il file da memorizzare è "lungo" n blocchi, si deve cercare all'interno della lista dello spazio libero se esiste una serie di n blocchi contigui. Ciò non è sempre possibile.
Questa operazione di ricerca di uno spazio idoneo può essere effettuata dal sistema operativo utilizzando diverse strategie:

- ricercando la prima area libera sufficientemente grande **(first-fit);**
- ricercando la prima area libera sufficientemente grande, iniziando l'operazione di scansione dal punto in cui si trova la testina **(next-fit);**
- ricercando l'area libera più piccola che possa contenere il file **(best-fit);**
- ricercando l'area libera più grande possibile **(worst-tit).**

Qualsiasi tecnica si utilizzi, però, sorgeranno comunque dei problemi: il disco continuerà a essere frammentato sempre di più, fino a trovarsi in una situazione che potrebbe sembrare

assurda, ma purtroppo è assolutamente reale: non sarà possibile memorizzare un file poiché non esisterà uno spazio contiguo abbastanza capiente da contenerlo, nonostante lo spazio libero complessivo sia sensibilmente maggiore di quello necessario.

Le operazioni consentite su un archivio a **organizzazione sequenziale** sono riassunte nel seguente diagramma:

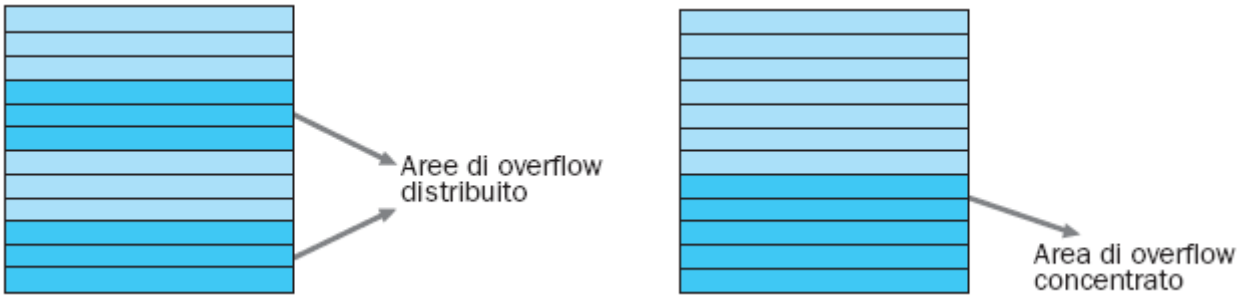


Inserimento

Per l'operazione di inserimento occorre distinguere sempre tra archivi non ordinati e archivi ordinati. Su un archivio non ordinato, l'inserimento non comporta problemi: il record si aggiunge in coda o, se nell'archivio esistono delle posizioni libere, può essere inserito nella prima di esse.

I problemi, questa volta, sorgono se l'archivio è ordinato. In questo caso, infatti, occorre individuare l'esatta posizione in cui collocare il record: appare evidente che, una volta individuata la posizione, occorrerà traslare tutti i record successivi in modo da creare lo spazio. Se nell'archivio ci fossero posizioni libere, l'inserimento sarebbe meno oneroso, in quanto potremmo limitare la traslazione soltanto fino al primo spazio libero. Considerato che la presenza di tali spazi agevola questa operazione, perché non crearli di proposito in

fase di progettazione dell'archivio? Operando in questo modo, occorre distribuire un congruo numero di aree libere lungo l'archivio, scegliendo opportunamente il numero e la dimensione in funzione del problema e delle risorse a disposizione.
È ovvio, comunque, che se la presenza di tali aree agevola l'inserimento, ostacola, però, la ricerca. Alternative efficienti possono essere applicate solo su archivi ordinati. Vediamone qualcuna.
Una prima soluzione consiste nel registrare i nuovi record in un archivio temporaneo o differenziale. Si procede poi, periodicamente, a una fusione tra i due archivi, così da rigenerare l'archivio principale. Questa soluzione non è delle migliori, in quanto alcune operazioni, come ad esempio la ricerca, sarebbero eseguite in modo poco efficiente perché, per reperire un record, occorrerebbe operare sia sull'archivio principale che su quello temporaneo, con conseguente aumento dei tempi di risposta.
La soluzione ottimale consiste nel predisporre nell'archivio delle aree libere, dette aree di overflow (o aree dei trabocchi), destinate ad accogliere i nuovi record inseriti. Queste aree possono essere disposte uniformemente nell'archivio principale o in un altro archivio differenziale (aree di overflow distribuito), oppure essere allocate in un unico punto dell'archivio o, ancora una volta, in un altro archivio differenziale (aree di overflow concentrato).



Aggiornamento (riscrittura)
La fase di aggiornamento, ossia l'operazione che consente di apportare modifiche ai campi di un record (a eccezione della chiave), può essere svolta, come al solito, in due modi diversi dipendenti dal metodo di accesso. Se l'accesso è sequenziale, l'aggiornamento di un record può essere svolto solo riscrivendo l'intero archivio. Le modifiche vengono raccolte in un altro archivio e, successivamente, si provvede all'aggiornamento dell'archivio principale. Nel caso di accesso diretto, invece, l'aggiornamento è una delle operazioni più semplici da realizzare, in quanto consta solo di tre semplici fasi:

- **ricerca del record di chiave K che si intende aggiornare;**
- **modifica del record;**
- **riscrittura del record modificato nella stessa posizione**.

Cancellazione
Anche per la cancellazione valgono le medesime considerazioni svolte per l'aggiornamento. Pertanto, nel caso di archivi sequenziali ad accesso sequenziale l'unico modo per poter cancellare un record (e quindi mantenere la corrispondenza tra la struttura logica e quella fisica) è quello di riscrivere l'intero archivio privato del record stesso servendosi di un archivio di appoggio.
Nel caso di archivi sequenziali ad accesso diretto, si ovvia a questo increscioso inconveniente effettuando una cancellazione logica, ossia predisponendo un apposito campo in grado di contenere un valore la cui presenza indica, appunto, la cancellazione del record.

Periodicamente si provvederà alla compattazione dell'archivio, eliminando fisicamente i record marcati per la cancellazione (cancellazione fisica).

la ricerca

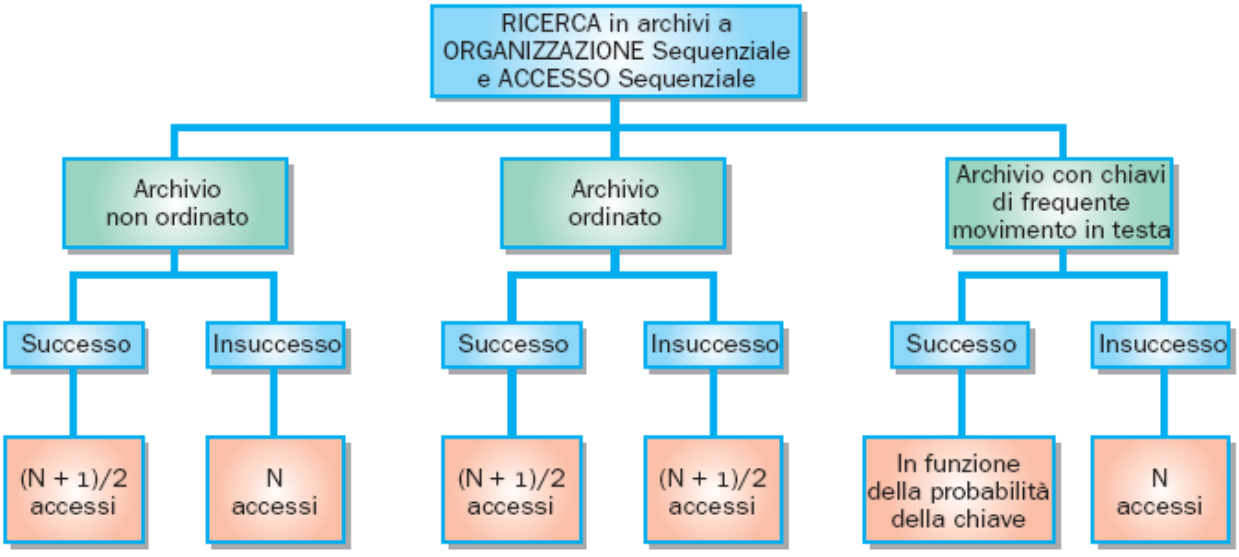
Le esigenze maggiormente diffuse nell'uso degli archivi sono quelle di *inserimento*, *consultazione* e *aggiornamento* di record. In presenza di tali esigenze, la ricerca dei record registrati è una delle operazioni più importanti e più frequenti. È ovvio, quindi, che tale operazione debba essere compiuta utilizzando una tecnica che consenta di ottenere il miglior risultato nel minor tempo possibile. La scelta del metodo riveste un ruolo di fondamentale importanza e varia in base al tipo di supporto di memoria utilizzato e alla presenza o meno di un ordinamento. In passato gli archivi venivano memorizzati sui nastri magnetici e, su di essi, l'unico metodo di ricerca attuabile era quello della ricerca completa dell'archivio, sino al reperimento del record di chiave desiderata. Il numero medio di accessi di tale metodo è il seguente:

• *successo*: (*N* + 1)/2 accessi;
• *insuccesso*: *N* accessi;
dove *N* è il numero di record presenti nell'archivio.

Quando si cominciò a trattare con archivi in cui si interveniva con frequenza solo su alcuni record, si cercò di ridurre il numero medio di accessi collocando tali record nelle prime posizioni dell'archivio. Questo metodo apportò un parziale incremento di efficienza solo in caso di successo: infatti, più probabile era la chiave, meno accessi erano necessari. In caso di insuccesso, invece, la situazione non mutava: erano sempre necessari *N* accessi. Con gli *archivi ordinati* si poté ricorrere a un altro metodo di ricerca, la *ricerca sequenziale*, ottenuta mediante l'ottimizzazione della scansione (la ricerca, infatti, si interrompe quando si trova la chiave cercata oppure quando ci si posiziona su un record di chiave maggiore di quella cercata). In tal caso, il numero medio di accessi diveniva:

• *successo*: (*N* + 1)/2 accessi;
• *insuccesso*: (*N* + 1)/2 accessi.

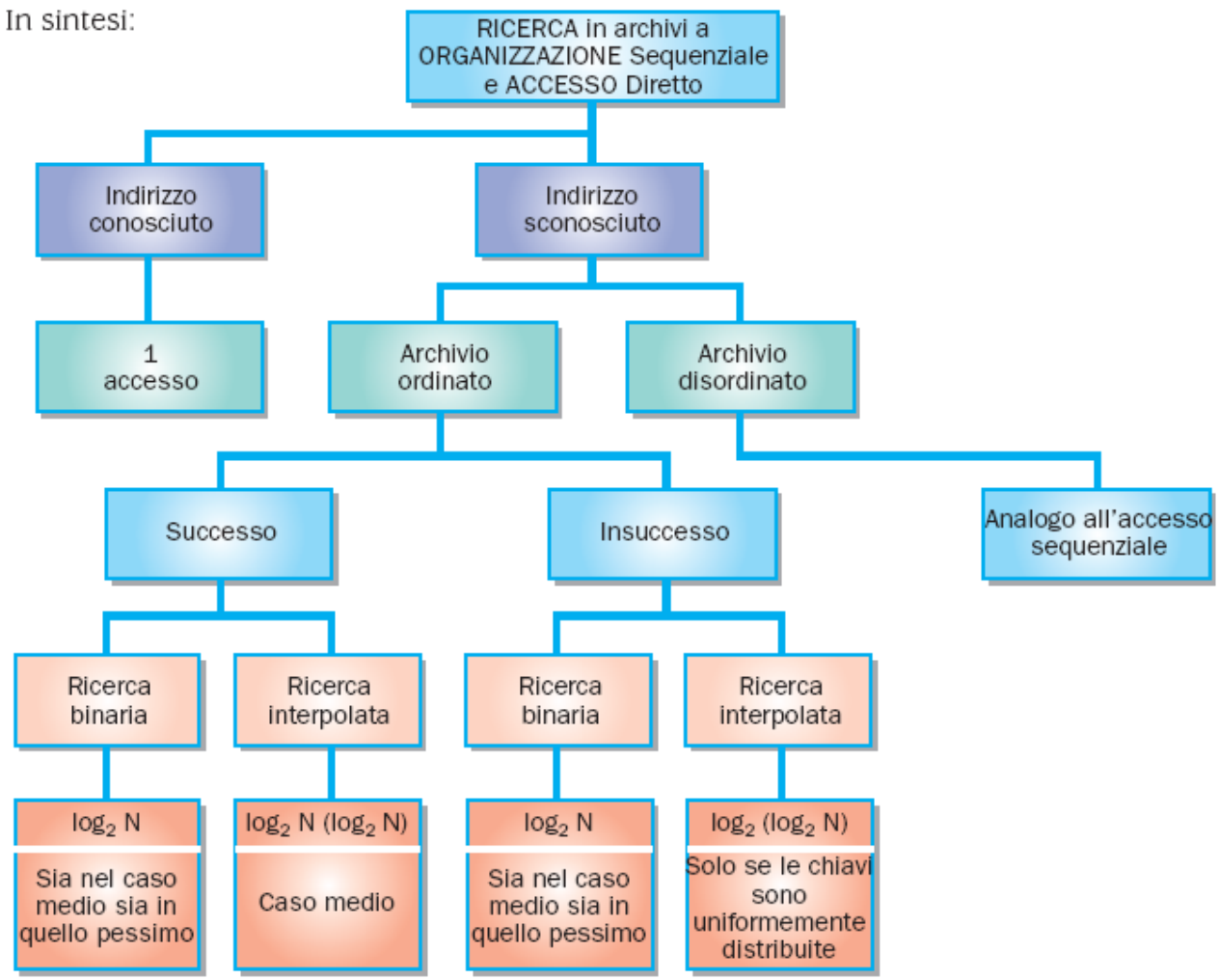
Ricapitoliamo:



Analizziamo, ora, il problema della ricerca applicato ad archivi a organizzazione sequenziale memorizzati su supporti ad accesso diretto. Quando è consentito l'*accesso diretto*, si possono verificare le seguenti situazioni:

- se si conosce l'indirizzo del record da ricercare, il problema non sussiste: mediante un accesso, ci si posiziona direttamente sul record;
- se l'indirizzo del record *non è noto* e *l'archivio è disordinato*, la situazione resta analoga a quella descritta per l'accesso sequenziale;
- se l'indirizzo del record *non è noto*, ma *l'archivio è ordinato*, il problema viene risolto con l'impiego di tecniche di ricerca molto efficienti, quali la *ricerca binaria*.

Per quanto riguarda questo metodo di ricerca, sia in caso di insuccesso che di successo (sia riferito al caso medio che a quello pessimo) si può dimostrare che il numero di accessi è pari a log(*N*). Il numero medio di accessi della *ricerca interpolata*, invece, è pari a log(log(*N*)), quindi è nettamente inferiore a quello della *ricerca binaria*.



File ad accesso diretto

Un file si dice organizzato in modo diretto quando è possibile accedere ad un singolo record specificato senza dover scandire tutto il file; non è quindi necessario memorizzare i records in modo contiguo.
In questo tipo di organizzazione è necessario che un particolare campo del record assuma il ruolo di chiave univoca in modo tale da rendere possibile, l'accesso al record corrispondente al file.

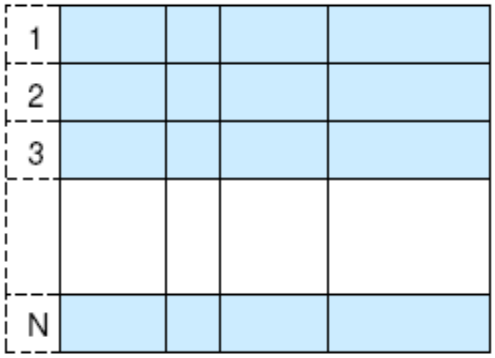
Un'organizzazione direct può essere classificata in:

- **organizzazione assoluta,**
- **organizzazione relativa (relative),**
- **organizzazione con indice(indexed).**

Nell'organizzazione assoluta o immediata il campo chiave che individua ciascun record è una chiave (primaria) che determina in modo univoco sia il record del file sia l'indirizzo fisico in cui tale record è memorizzato.
La struttura di questo tipo di file è simile a quella di un vettore, in quanto la lunghezza dei records deve essere fissata, ciò provoca un conseguente spreco di memoria anche se permette un accesso veloce al disco.
In tale tipo di organizzazione è il programmatore a doversi preoccupare della gestione della corrispondenza tra chiave logica del file e indirizzo corrispondente sul dispositivo esterno di memoria, sia in fase di lettura che di scrittura di un record.
Per tale motivo è raramente usato.

Organizzazione Relativa

In un archivio **casuale relativo**, o ad **accesso diretto** è possibile accedere ai record, per memorizzarli e/o per leggerli, direttamente, semplicemente specificando qual' è la loro posizione relativamente all'inizio dell'archivio stesso.



L'apertura dell'archivio è unica (non occorre specificare se in input, in output o in append, come per gli archivi sequenziali), dopo di che le istruzioni che si utilizzeranno sono del tipo: *lettura record **n***, *scrittura record **n*** (dove "***n***" è il numero del record da leggere e/o scrivere).

Nell'organizzazione relative, la gestione degli indirizzi fisici è affidata al sistema operativo. Per determinare l'indirizzo del k-esimo record, il sistema operativo moltiplica la dimensione del record, ossia la sua lunghezza, per (K–1) e, successivamente,somma tale valore a quello dell'indirizzo fisico del primo record registrato nell'archivio.

Tra i **principali vantaggi** di questo tipo di organizzazione, ricordiamo che:

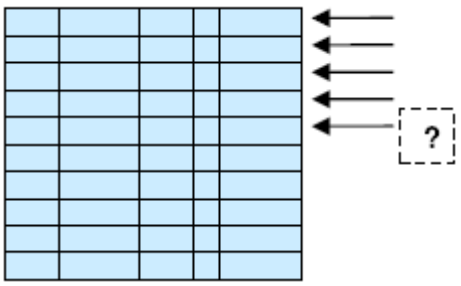
- a ogni record corrisponde una chiave numerica indicante la posizione occupata dal record all'interno dell'archivio (indirizzo logico);
- l'organizzazione relative può essere utilizzata solo su supporti di memorizzazione ausiliari ad accesso diretto;
- in tale tipo di organizzazione è consentito sia l'accesso sequenziale, sia quello diretto, quindi l'organizzazione relative può essere gestita indifferentemente come archivio sequenziale o come archivio non sequenziale;
- 'organizzazione relative risulta molto vantaggiosa per le operazioni interattive;
- utilizzando puntatori in questo tipo di organizzazione è possibile ottenere un ordine logico diverso da quello fisico;
- l'organizzazione relative offre la possibilità di accedere ai record con una chiave alfanumerica (ma ciò è a carico del programmatore).

Tra gli **svantaggi**, invece, ricordiamo che gli archivi relative presentano un inconveniente: spesso è molto difficile riuscire a impostare una relazione univoca tra la chiave primaria e la posizione nella quale memorizzare il record.
Questo inconveniente obbliga il programmatore a utilizzare chiavi legate strettamente alla posizione relativa del singolo record all'interno dell'archivio, oppure a utilizzare particolari metodi di randomizzazione.

Nell'organizzazione relative la **cancellazione** può avvenire solo logicamente, a eccezione dell'ultimo record.
Per **modificare** uno o più campi del record basta leggere il record, modificare i valori e riscriverlo.
Con questa organizzazione non si ha l'EOF, come nel sequenziale, e si possono avere record vuoti.
Oltre all'*accesso diretto*, specificando il numero del record, è possibile anche l'*accesso sequenziale* (inserendo, ad esempio, le istruzioni di lettura/scrittura record all'interno di un ciclo *for-next* con indice da 1 ad N, indice che equivale al numero del record da leggere/scrivere).

Problemi di ricerca

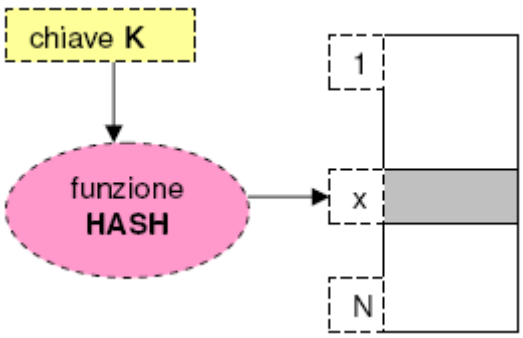
L'organizzazione casuale è ottima quando si conosce a priori la posizione del record da leggere e/o da scrivere, quando cioè la chiave di accesso al record è un numero intero da 1 a N.
Le cose, però, si complicano quando la chiave di accesso al record non è la sua posizione. Ad esempio in un archivio anagrafico la chiave di accesso potrebbe essere **Cognome+Nome** (chiave alfanumerica).In tal caso si pone il problema della ricerca del record.



Un primo approccio, il più semplice dal punto di vista della programmazione, è la ***ricerca sequenziale*** (o ***scansione lineare***):
si comincia a leggere dal primo record, sequenzialmente, fino a che non si trova ciò che si cerca, se esiste, o fino alla fine dell'archivio, se non esiste.
Con questo tipo di ricerca, su un archivio con N record, il numero medio di letture è **N/2** (in caso di successo, perché in caso di insuccesso tale media peggiora notevolmente in quanto, per poter dire che un dato record non esiste, occorre leggere tutti gli N record dell'archivio).
Se la velocità di ricerca è prioritaria, si può usare un altro tipo di ricerca, cioè la ***ricerca binaria*** o ***dicotomica***. Questo tipo di ricerca presuppone che l'archivio sia fisicamente ***ordinato*** sul campo di ricerca (ad esempio il Cognome+Nome) e che non abbia record vuoti.

Per accedere direttamente al record fornendo il valore di una chiave alfanumerica. Su tale chiave viene applicata una funzione di trasformazione - **funzione HASH** - che "trasforma" la chiave alfanumerica in un numero intero (che è il numero di record).
Tale funzione dovrebbe essere tale che:

- **f(Ki) = R (con R ∈ [1,N])**
- **f(Ki) ≠ f(Kj) (per ogni Ki ≠ Kj)**
- **f completa (copre da 1 a N)**



In pratica una tale funzione - **funzione HASH perfetta** - non esiste, se non per trasformazioni banali quali K $\in$ [A,B,C,…,Z] e R $\in$ [1÷26].
Ad esempio, con una chiave di 5 caratteri alfabetici, per avere una funzione HASH "perfetta", ci vorrebbe
un archivio con 26^5 = 11.881.376 record. Impensabile.
Un esempio di funzione HASH è **f(K) = integer(K) modulo M** (dove "integer(K)" è l'intero formato dai codici ASCII dei caratteri di K ed M è un numero primo).
Si accetta il fatto che due o più chiavi possano collidere e dare lo stesso numero di record. Occorre allora gestire tali **collisioni** (che si hanno quando f(Ki) = f(Kj) con Ki ≠ Kj). Esistono diverse tecniche di gestione delle collisioni: **scansione lineare**, **concatenazione**, **archivio delle collisioni**.
Con questa organizzazione si possono usare istruzioni di accesso diretto tipo leggi record con chiave K, e memorizza record con chiave K, ma non si possono reperire i record sequenzialmente in ordine di chiave, dalla più piccola alla più grande.
Originariamente la determinazione della funzione HASH di trasformazione della chiave e la gestione delle collisioni erano totalmente a carico del programmatore. In seguito il tutto è stato realizzato a livello dei metodi di accesso del File System del Sistema Operativo, sotto il nome di "accesso per chiave" o "index".

La scelta della *funzione hash* assume un ruolo di fondamentale importanza, in quanto una funzione di trasformazione non idonea può rendere inefficiente o anche completamente invalida l'organizzazione stessa. Una funzione hash ottimale deve:

1. essere facilmente calcolabile, cioè composta da calcoli che siano i più semplici possibile, in modo da non appesantire il procedimento di conversione della chiave e diminuire, così, il tempo di accesso;
2. produrre sempre lo stesso indirizzo a partire dalla stessa chiave;
3. generare indirizzi uniformemente distribuiti nell'ambito dell'archivio;
4. generare indirizzi casualmente distribuiti nell'ambito dell'archivio, relativamente a chiavi simili (esempi di chiavi simili possono essere X1, X2, X3, oppure DARE, MARE, CARE);
5. coprire l'intero intervallo degli indirizzi evitando, se possibile, che ci siano indirizzi che non vengono mai generati;
6. generare indirizzi diversi per chiavi diverse.

Relativamente all'ultimo punto, quindi, la funzione ideale è quella che a ogni valore della chiave primaria fa corrispondere sempre un indirizzo diverso, di modo che ogni record possa essere raggiunto tramite un solo accesso. Quando ciò accade si parla di trasformazione perfetta.

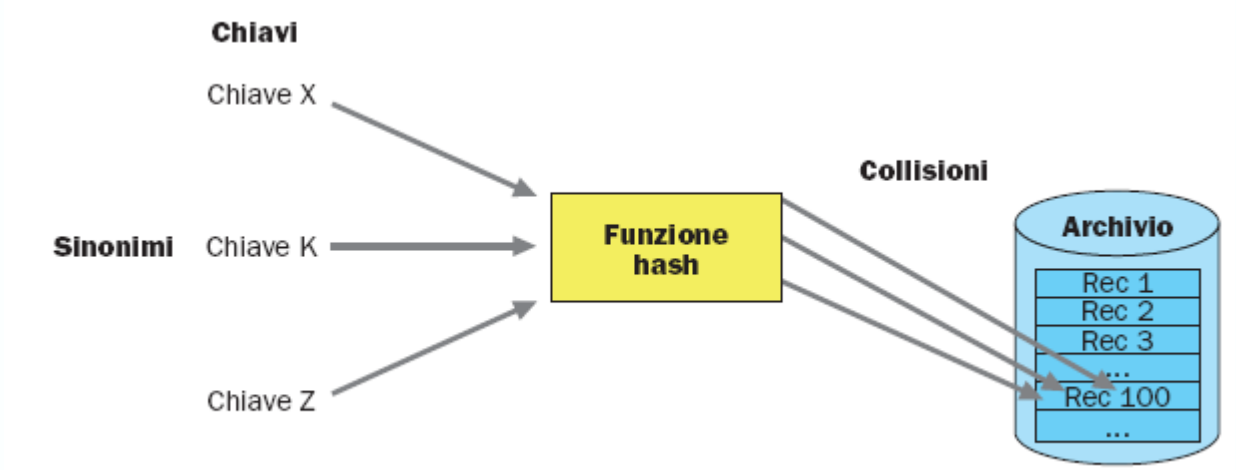
Supponiamo di dover memorizzare in un archivio i 1000 prodotti giacenti nel magazzino di un generico punto vendita, con un codice che varia tra 5000 e 5999. Supponiamo, inoltre, che il campo Codice Prodotto, al quale attribuiamo la funzione di chiave primaria, sia composto al massimo da quattro cifre. La funzione hash più semplice consiste nel sottrarre 5000 al valore di ogni codice e, successivamente, addizionare 1:

H(K) = H(Codice Prodotto) = (Codice Prodotto) – 5000) + 1

La funzione restituisce valori compresi tra 1 e 1000 facendo si che a ogni codice corrisponda un solo record.

La situazione presentata nell'esempio però, è ben lontana dalla realtà, in quanto è molto difficile realizzare funzioni hash che consentano una corrispondenza uno-a-uno tra chiave e indirizzo. Funzioni come quella dell'esempio, infatti, richiedono che il numero di record

che compongono l'archivio fisico sia esattamente uguale al numero dei possibili valori che la chiave può assumere.
Nella realtà, il numero di chiavi effettive da gestire è sensibilmente inferiore a quello delle chiavi ipotetiche e ciò porterebbe a un notevole spreco di memoria e, di conseguenza, a un'inefficienza generale dell'intera organizzazione.
Per questi motivi si opta per altre funzioni hash che, pur non garantendo una corrispondenza uno-a-uno, cioè una trasformazione perfetta, non si discostano molto dal modello ideale. La funzione, in tal caso, produce una corrispondenza molti a-uno, cioè accade che due o più chiavi, in fase di trasformazione, possano generare lo stesso indirizzo.



Quando ciò accade si produce una collisione, per risolvere la quale occorre disporre di particolari procedure che collochino le registrazioni in conflitto, dette sinonimi, in record di indirizzo diverso. I requisiti descritti precedentemente per la scelta della funzione hash al punto 4 e al punto 5 rappresentano la condizione necessaria per contenere il fenomeno della collisione.

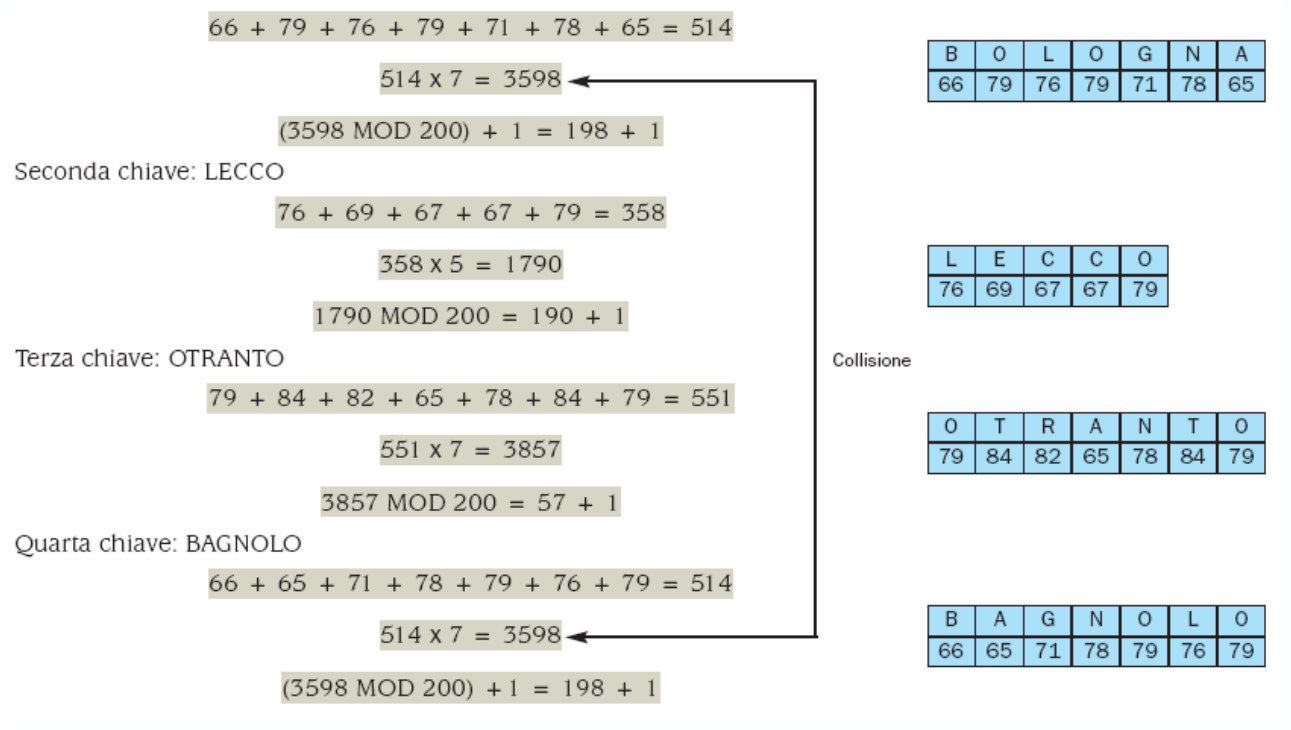
A tal proposito, possiamo anche formulare un settimo requisito dicendo che scegliere una funzione hash ottimale significa scegliere una funzione che:

7. generi il minor numero possibile di collisioni e definisca apposite procedure per l'allocazione dei sinonimi (gestione delle collisioni).

Supponiamo di dover stabilire l'indirizzo logico corrispondente a record aventi le seguenti chiavi (BOLOGNA, LECCO, OTRANTO, BAGNOLO) per un archivio composto al massimo da 200 record. Per allocare le suddette chiavi ci serviamo del seguente metodo:

1. si associa a ogni carattere della chiave il corrispondente valore ASCII;
2. si sommano tutti i valori ottenuti;
3. si moltiplica tale somma per il numero di cifre che compongono il nome della chiave;
4. si calcola il resto della divisione del numero ottenuto per il numero di record inseribili, cioè 200, e si somma + 1, ottenendo il valore dell'indirizzo.

Cominciamo con la prima chiave: BOLOGNA

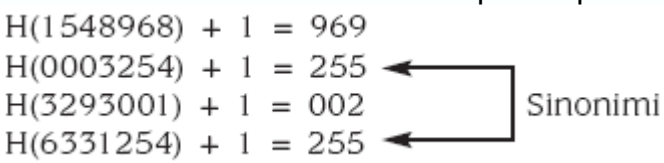


L'organizzazione hash calcolo degli indirizzi

Assodato che è quasi impossibile realizzare funzioni che consentano una trasformazione perfetta e che, di conseguenza, il problema delle collisioni sarà sempre presente (indipendentemente dalla funzione), occorre concentrare le nostre forze sullo sviluppo di funzioni efficienti, piuttosto che perfette. Esistono molti metodi per calcolare gli indirizzi; esaminiamone solo alcuni.

Il metodo del troncamento

Questo metodo viene utilizzato quando la chiave numerica è composta da un elevato numero di cifre e il numero di record che possono essere memorizzati è abbastanza limitato. Consiste nell'estrarre dalla chiave un sottoinsieme di cifre che rappresentano l'indirizzo.
Supponiamo di avere un archivio che contiene informazioni sui tesserati di una palestra e sul quale possono essere memorizzati 1000 record partendo dall'indirizzo 1.
Il ruolo di chiave primaria viene attribuito al numero di tessera di ogni socio, che è composto da un numero di sette cifre. Dobbiamo costruire una corrispondenza che a ogni valore della chiave di sette cifre associ un indirizzo di tre cifre, ossia da 000 a 999. La funzione hash, pertanto, sarà ottenuta eliminando dalla chiave le prime quattro cifre e sommando 1:



Questo semplicissimo metodo è molto efficiente nel caso in cui le chiavi numeriche siano essenzialmente consecutive e con poche interruzioni. Gli svantaggi di tale funzione risiedono nel fatto che essa produce sinonimi ogni volta che si incontrano chiavi aventi le ultime tre cifre uguali e che il numero di indirizzi logici a disposizione deve essere sempre una potenza di 10. Ciò richiede (come spesso accade) che l'archivio debba essere composto da un numero di record superiore a quello effettivamente necessario. Riferiamoci all'esempio precedente: se la palestra ha solo 130 soci, deve prevedere sempre un archi-

vio composto da 1000: è utile sovradimensionare l'archivio (magari prevederne 160), ma predisporre 1000 record è, ovviamente, un enorme spreco di memoria! Applichiamo questo metodo a una chiave alfanumerica. Supponiamo che la chiave da trasformare debba essere composta da quattro caratteri ottenuti estraendo gli ultimi quattro caratteri della chiave primaria. Per procedere alla trasformazione, si può utilizzare il semplicissimo metodo che pone in corrispondenza le 26 lettere dell'alfabeto con i primi 26 numeri naturali, proprio come riportato nella tabella riportata a lato.
A questo punto, proprio come nei sistemi di numerazione pesati, si procede alla costruzione di una funzione che calcoli la sommatoria dei prodotti di ogni carattere per il suo peso, ossia per 26 elevato alla posizione occupata all'interno della parola. Proviamo a calcolare l'indirizzo, o meglio, il numero corrispondente alla chiave HELLOCIAO. Estraiamo gli ultimi quattro caratteri dalla chiave HELLOCIAO:

$$\begin{aligned} CIAO &= C \times 26^3 + I \times 26^2 + A \times 26^1 + O \times 26^0 = \\ &= 2 \times 26^3 + 8 \times 26^2 + 0 \times 26^1 + 14 \times 26^0 = \\ &= 2 \times 17576 + 8 \times 676 + 0 + 14 \times 1 = \\ &= 35152 + 5408 + 14 = \\ &= 40574 \end{aligned}$$

A	0
B	1
C	2
...	...
...	...
Z	25

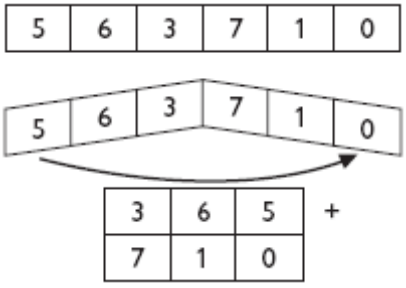
A questo punto, per calcolare il reale indirizzo, si può applicare il metodo del troncamento, ma, per farlo, il programmatore deve aver stabilito a priori qual è il numero massimo di record che può contenere l'archivio, in modo da poterlo approssimare in modo migliore. Come per il caso di una chiave numerica, anche in questo caso lo svantaggio principale deriva da un notevole spreco di memoria e dalla possibilità di generare diverse collisioni, visto che esistono diverse parole che possono iniziare o terminare con le stesse lettere.

Il metodo folding

Questo metodo consiste nel dividere la chiave numerica in due o più parti. Successivamente, tali parti vengono sommate tra loro e il valore ottenuto (o una sua parte), viene utilizzato come indirizzo. Il nome folding (termine inglese che significa "piegare") si spiega se si immagina di piegare la chiave congiungendo le sue estremità, proprio come se fosse scritta su un foglio di carta, e di sommare tra loro le cifre che ricadono nella stessa posizione. Se la somma ottenuta supera il numero massimo di cifre di cui deve essere composto l'indirizzo, si procede al troncamento delle prime cifre.
Supponiamo che la chiave sia 563710:

H(563710) = 365 + 710 = 1075

Graficamente abbiamo quanto è riportato a lato. Il valore dell'indirizzo è, quindi, 1075. Anche questo metodo genera diverse collisioni, poiché, ad esempio, H(563710) = H(349132). Inoltre, presenta gli stessi svantaggi del metodo di troncamento poiché, anche in questo caso, all'indirizzo viene riservato uno spazio pari a una potenza di 10. Ribadiamo, infine, che la scelta di suddividere la chiave in due o più parti è in relazione alla specifica applicazione. Tale suddivisione deve essere attentamente valutata, al fine di con-



sentire una distribuzione uniforme degli indirizzi, per ridurre il più possibile il numero delle collisioni.

Il metodo della divisione modulo N

Con questo metodo l'indirizzo del record si ottiene dal resto della divisione della chiave numerica per il numero massimo di record memorizzabili nell'archivio. In generale, se indichiamo con k la chiave e con N il numero massimo di record, l'indirizzo del record è dato da: **H(K) = K MOD N**

Sappiamo che l'operatore modulo N fornisce sempre valori compresi tra **0 ed N – 1**. Se vogliamo che la funzione restituisca indirizzi Hash compresi tra 1 ed *N* dobbiamo apportare alla funzione la seguente modifica: **H(K) = (K MOD N) + 1**

Anche questo metodo ha generato dei sinonimi, ma questo metodo presenta il vantaggio di poter essere applicato anche in presenza di un numero massimo di record che non è multiplo di 10. Gli indirizzi generati, proprio per la definizione di modulo, non necessitano di troncamento. Per quanto riguarda i sinonimi, la scelta del valore N del modulo è il punto fondamentale per ridurre al minimo la loro presenza.

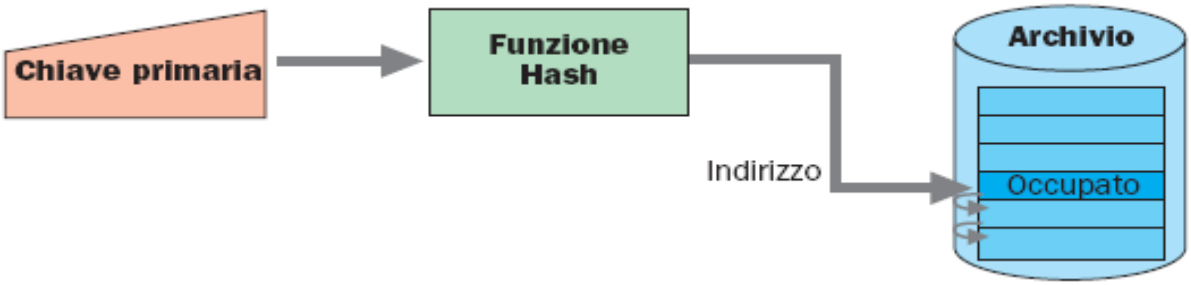
Supponiamo di dover memorizzare le seguenti dieci chiavi in un archivio composto, al massimo, da 20 record: **92 64 21 80 12 13 9 88 11 58**.

Chiave K	H(K) = (K mod 20) + 1
92	12 + 1 = 13
64	4 + 1 = 5
21	1 + 1 = 2
80	0 + 1 = 1
12	12 + 1 = 13
13	13 + 1 = 14
9	9 + 1 = 10
88	8 + 1 = 9
11	11 + 1 = 12
58	18 + 1 = 19

Gestione delle Collisioni

Scansione lineare o indirizzamento aperto

Quando la funzione hash produce un indirizzo già occupato da un altro record, occorre effettuare la scansione dell'archivio in modo da ricercare il primo posto libero in cui sistemare la chiave che ha generato la collisione.



La tecnica della scansione lineare, detta anche dell'indirizzamento aperto, consiste proprio in questo. Se, ad esempio, la collisione avviene sull'indirizzo 15 e poniamo come passo di scansione 3, occorre esaminare i record di indirizzo 18, 21, 24, 27 sino al reperimento di un posto vuoto. Se, in particolare, p = 1 si parla di scansione lineare con passo unitario. La scelta del passo unitario assicura il controllo di tutti gli indirizzi, in quanto, se la chiave col-

lide su un indirizzo, si va a controllare la posizione immediatamente successiva, quella successiva ancora e così via. Il processo termina quando si trova un posto libero, oppure quando è stata esaminata metà delle posizioni presenti all'interno dell'archivio (in tal caso viene comunicato un messaggio di archivio pieno). La scansione non si interrompe quando si incontra la fine dell'archivio, ma riprende dalla prima posizione.
Questo metodo presenta un inconveniente, detto agglomerazione o addensamento primario delle chiavi (***clustering***), cioè la presenza di aree di memoria (agglomerati o ***cluster***) nelle quali confluiscono varie sequenze di sinonimi, che presentano indirizzi coincidenti da un certo punto in poi (creando lunghe catene di record adiacenti, con conseguente significativo rallentamento del tempo di accesso ai record).
Supponiamo di memorizzare le seguenti chiavi in un archivio composto al massimo da 9 record: 9 28 29 35 24 48 71 64
Per generare gli indirizzi ci serviamo del metodo della divisione modulo 9. Abbiamo:

```
9  → (9 mod 9)  + 1 = 0 + 1 = 1  la chiave 9 viene memorizzata in posizione 1
28 → (28 mod 9) + 1 = 1 + 1 = 2  la chiave 28 viene memorizzata in posizione 2
29 → (29 mod 9) + 1 = 2 + 1 = 3  la chiave 29 viene memorizzata in posizione 3
35 → (35 mod 9) + 1 = 8 + 1 = 9  la chiave 35 viene memorizzata in posizione 9
24 → (24 mod 9) + 1 = 6 + 1 = 7  la chiave 24 viene memorizzata in posizione 7
48 → (48 mod 9) + 1 = 3 + 1 = 4  la chiave 48 viene memorizzata in posizione 4
71 → (71 mod 9) + 1 = 8 + 1 = 9  collisione con la chiave 35.
```

A questo punto, in base alla tecnica della scansione lineare con passo unitario, dobbiamo postarci sul record successivo per vedere se la posizione è libera: dobbiamo andare sulla posizione 10. Ciò, ovviamente, non è realizzabile in quanto abbiamo supposto che l'archivio sia composto al massimo da 9 record. Occorre, pertanto, applicare la divisione anche all'indirizzo prodotto. Abbiamo:

```
71 → (71 mod 9) + 1 = 8 + 1 = 9  collisione con la chiave 35
     (9 mod 9)  + 1 = 0 + 1 = 1  collisione con la chiave 9
     (1 mod 9)  + 1 = 1 + 1 = 2  collisione con la chiave 28
     (2 mod 9)  + 1 = 2 + 1 = 3  collisione con la chiave 29
     (3 mod 9)  + 1 = 3 + 1 = 4  collisione con la chiave 48
     (4 mod 9)  + 1 = 4 + 1 = 5  la chiave 71 viene memorizzata in posizione 5.
```

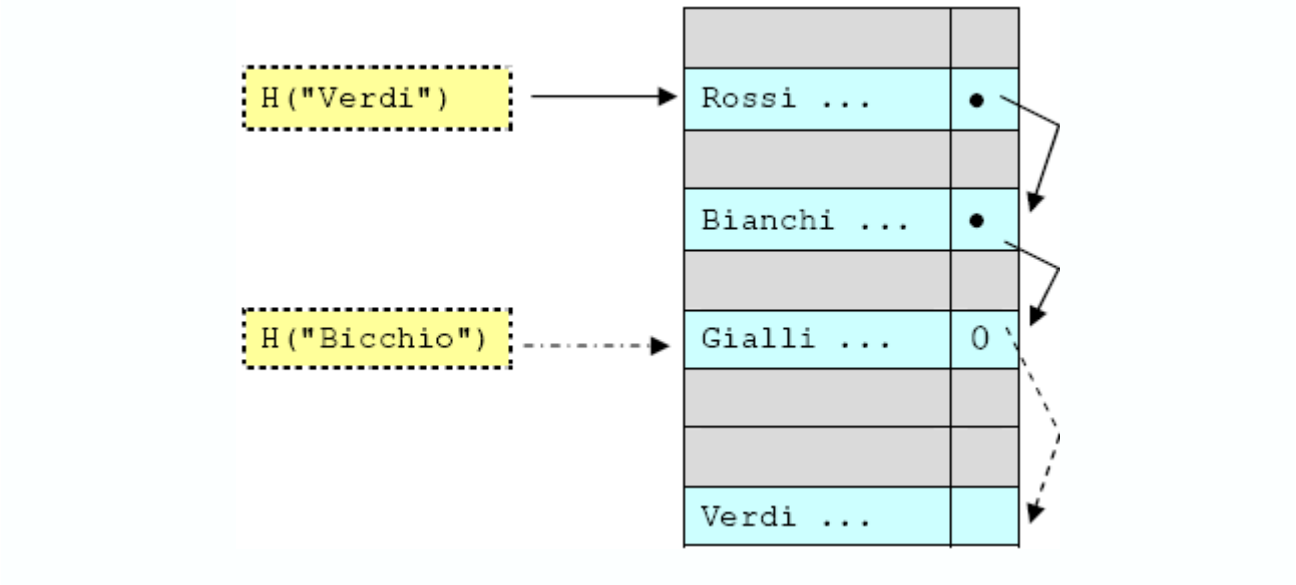
Ora analizziamo l'ultima chiave:

```
64 → (64 mod 9) + 1 = 1 + 1 = 2  collisione con la chiave 28.
```

Poiché l'indirizzo 2 è interno all'intervallo, si cominciano a esaminare una a una le successive posizioni, sino a incontrare un posto libero.
Vengono quindi scandite le posizioni 3, 4, 5 e, infine, la posizione 6, sulla quale la chiave 64 viene memorizzata.
Come si può constatare, la chiave 64 ha dovuto attraversare una parte della catena precedente, anche se i record individuati da questi indirizzi erano occupati.

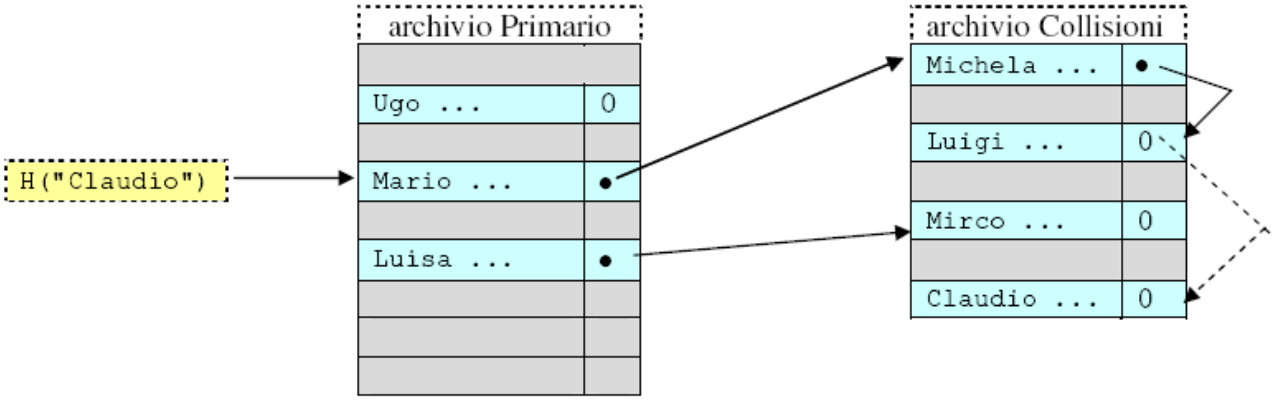
Catene confluenti

In caso di collisione si cerca un record libero per il suo inserimento e lo si concatena alla lista di collisione intercettata. Le collisioni occuperanno record riservati ad altre chiavi, aumentando la possibilità di successive collisioni e la lista delle collisioni intercettata. Quindi in un'unica lista, confluiscono le collisioni di tutte le chiavi che fanno parte della lista stessa. Da qui *catene confluenti*. La lettura comunque è più veloce in quanto si leggono al massimo solo gli elementi della lista. I record liberi sono legati a lista



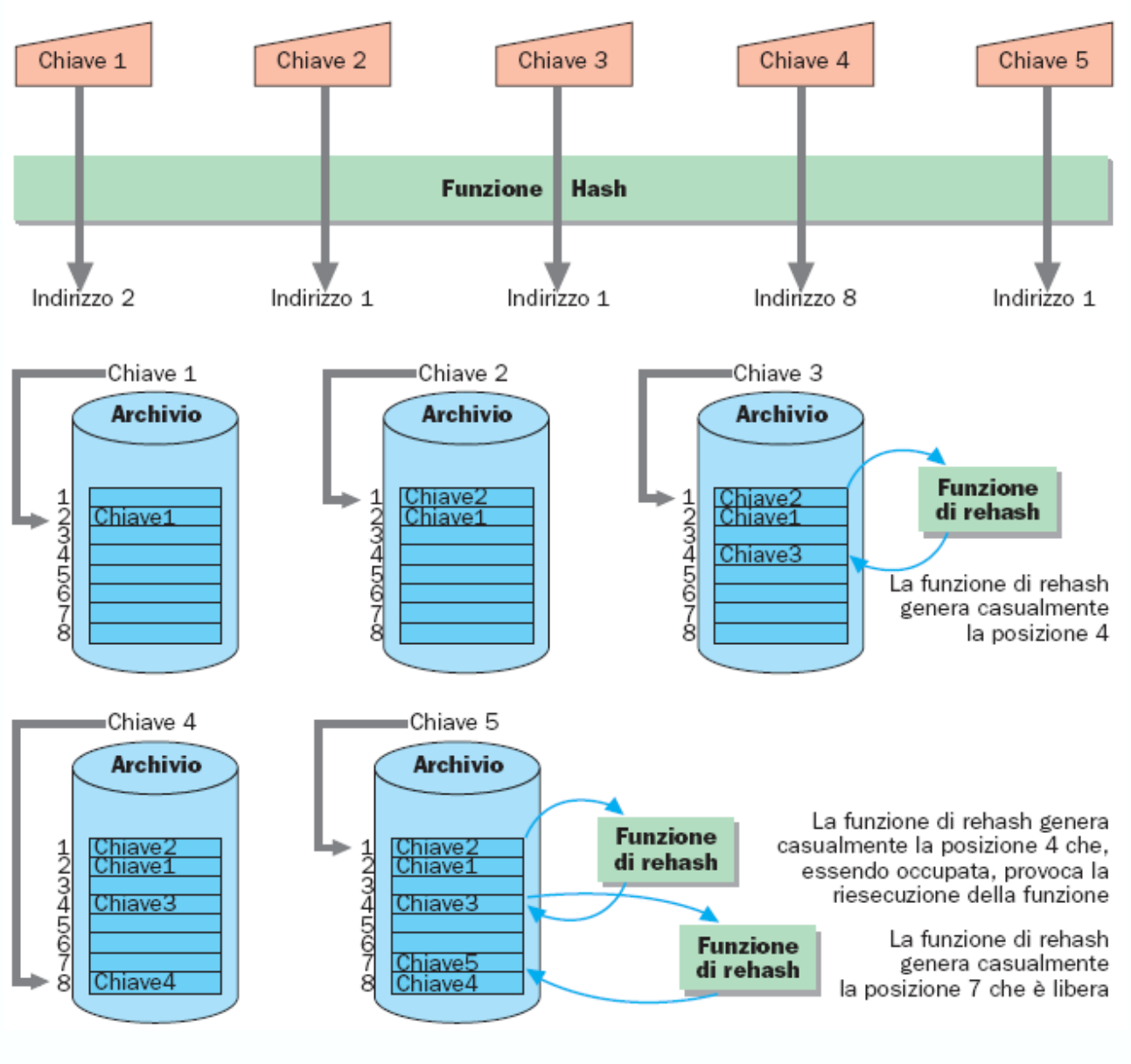
Archivio delle Collisioni

Tutte le collisioni sono memorizzate in un altro archivio e legate a lista. Ogni lista contiene solo le collisioni di un dato record nell'archivio primario. Questa tecnica, che richiede l'uso di un altro archivio e la gestione delle liste, è la più efficiente delle tre illustrate.



Scansione non lineare o metodo pseudo-random

Il metodo che stiamo per descrivere rappresenta una variante dell'indirizzamento aperto e, come tale, nasce anch'esso allo scopo di eliminare o di tamponare il problema dell'agglomerazione. Per farlo, questo metodo ricorre a una tecnica di memorizzazione che non prevede una scansione sequenziale dell'archivio, poiché, invece di utilizzare il passo di scansione, genera un numero casuale (o, meglio, pseudocasuale) ogniqualvolta si verifica una collisione. Tale metodo si definisce scansione non lineare o ***pseudo-random***. In altri termini, si calcola l'indirizzo per iniziare la scansione: se su tale indirizzo si verifica una collisione, non ci si sposta nella posizione immediatamente successiva bensì si ricalcola un nuovo indirizzo, detto indirizzo di rehash, servendosi di una nuova funzione, detta funzione di rehash. L'indirizzo di rehash può essere calcolato con una qualsiasi funzione che generi numeri pseudocasuali e che, per ogni passo, definisca un incremento differente per l'indirizzo. Circa la casualità, ribadiamo che l'indirizzo di rehash deve essere necessariamente pseudocasuale e non puramente casuale, in quanto, in fase di ricerca di una chiave, devono essere ricalcolati gli stessi indirizzi prodotti per la memorizzazione, ossia deve essere ripetuta la stessa sequenza di indirizzi.


Chiave 1
Chiave 2
Chiave 3
Chiave 4
Chiave 5
Funzione Hash
Indirizzo 2
Indirizzo 1
Indirizzo 1
Indirizzo 8
Indirizzo 1
Archivio
Chiave1
Chiave2
Chiave3
Chiave4
Chiave5
Funzione di rehash
La funzione di rehash genera casualmente la posizione 4
La funzione di rehash genera casualmente la posizione 4 che, essendo occupata, provoca la riesecuzione della funzione
La funzione di rehash genera casualmente la posizione 7 che è libera

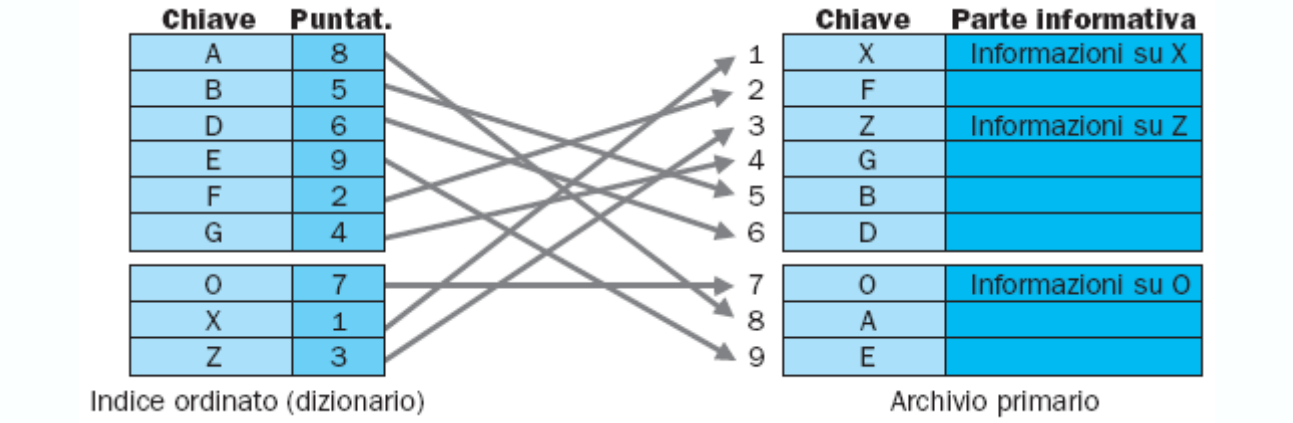
Organizzazione SEQUENZIALE con INDICE

Gli archivi registrati su supporti fisici che permettono l'accesso diretto possono essere gestiti anche con una variante dell'organizzazione sequenziale, caratterizzata dalla possibilità di velocizzare la ricerca di un record mediante uno o più *indici*.

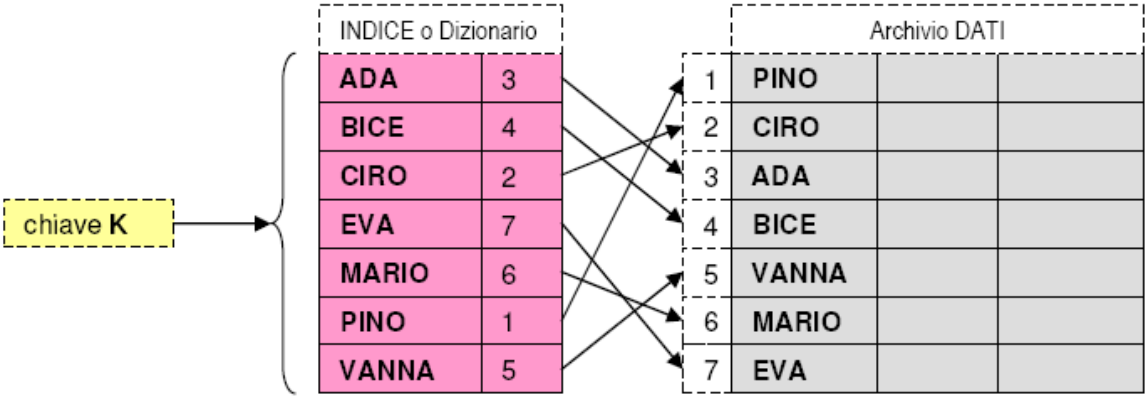
Nell'**organizzazione sequenziale a indici** (*indexed sequential*) si possono distinguere due elementi fondamentali:

- un **archivio primario**, caratterizzato da record consecutivi che possono anche essere ordinati;
- un **archivio secondario** o *indice ordinato* (**dizionario**) o una gerarchia di indici, i cui elementi sono composti, generalmente, da due campi:
  - un **campo chiave**, contenente la chiave del record;
  - un **campo puntatore**, contenente la posizione del record all'interno dell'archivio primario.

In un archivio **sequenziale con indice** (o **index sequential**) si accede al record fornendo il valore di ua chiave alfanumerica (come è anche possibile per l'organizzazione **a indici** precedentemente esaminata), ma è anche possibile l'accesso sequenziale, seguendo l'ordinamento della chiave di accesso. La chiave di accesso non viene "trasformata" come nell'organizzazione a indici, ma viene gestita in una struttura a parte, detta **Indice** o **Dizionario**, che contiene tutte le ***chiavi logicamente ordinate*** (in ordine crescente o decrescente) e, per ciascuna chiave, contiene il riferimento (puntatore) al relativo record nell'**Archivio Dati**. Tale chiave individua univocamente una registrazione ed è detta **Chiave Primaria**.



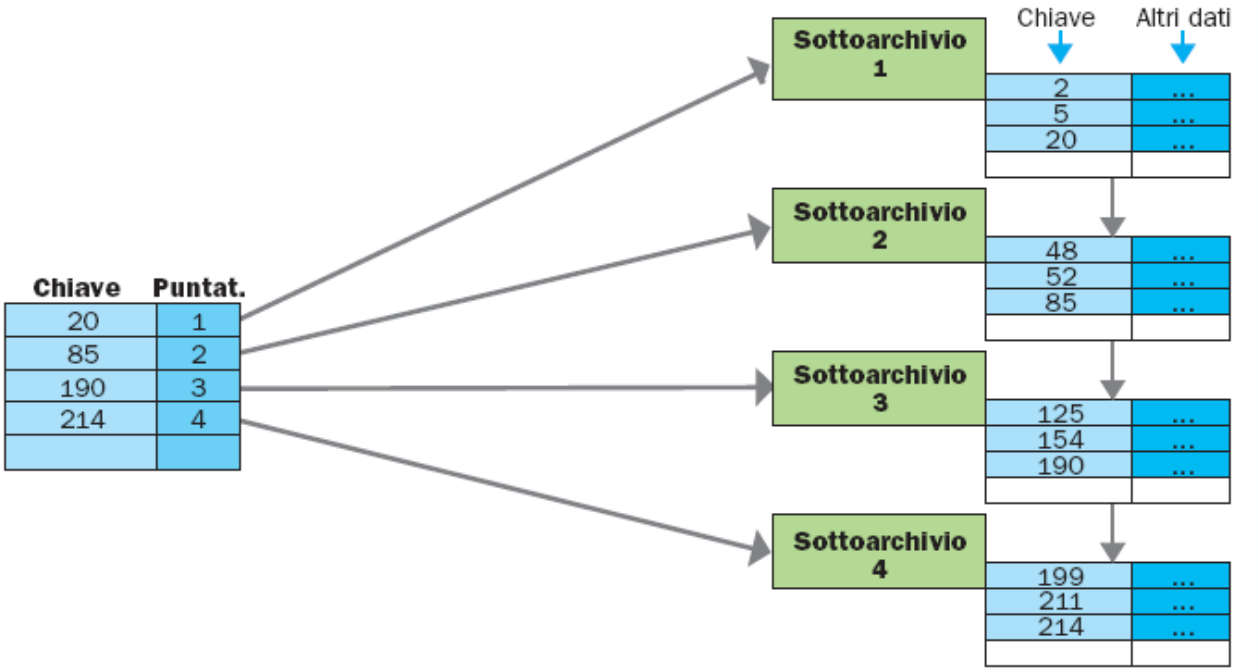
Esempio:



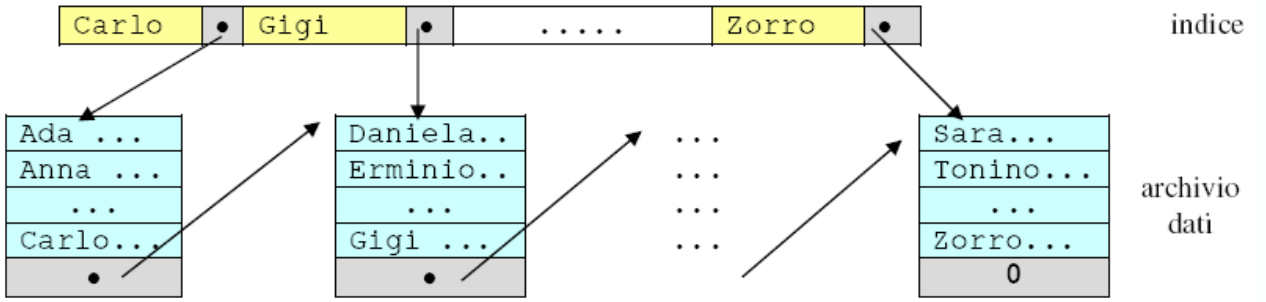
Questa soluzione è dispendiosa sia in termini di occupazione di memoria, sia di gestione delle informazioni. Per sfruttare la totalità dei vantaggi offerti da questo tipo di organizza-

zione, e in particolare potenziare le operazioni di ricerca dei record, occorre innanzitutto distinguere fra:
• strutture sequenziali con indice *ordinate*;
• strutture sequenziali con indice *non ordinate*;
e tenere conto del metodo di costituzione dell'indice.

Nelle strutture ordinate, l'archivio primario viene implementato in una *struttura a pagine* (o *sottoarchivi* che sono costituiti da blocchi contigui con ugual numero di record) e l'indirizzo della chiave più alta di ciascun sottoarchivio è contenuto nell'indice i cui record sono del tipo: (*Kh*, *P*) dove *Kh* indica la chiave più alta e *P* il numero di sottoarchivio.
In questo modo, quindi, l'indice viene utilizzato esclusivamente per la ricerca dei vari sottoarchivi. Analizziamo la seguente figura:



In pratica, il valore della chiave nell'indice indica esplicitamente la chiave di valore più alto contenuta all'interno del sottoarchivio, mentre il puntatore indica implicitamente la posizione della chiave più bassa poiché rappresenta l'indirizzo del sottoarchivio e, quindi, il primo record presente, cioè quello con la chiave di valore più basso.
Esempio:



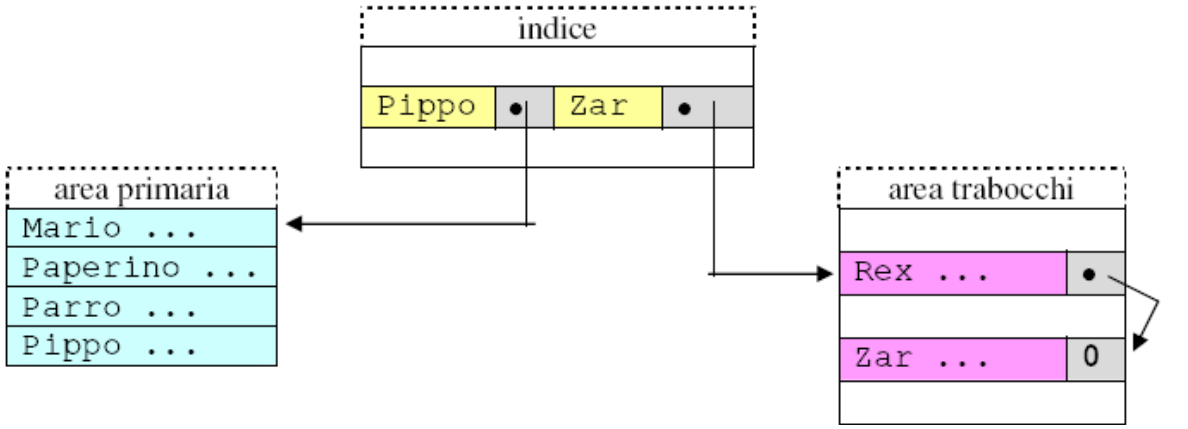
Generalmente, la ricerca di una chiave *K* avviene rispettando la seguente procedura:

• ricerca nell'indice la prima chiave *Kh* ≥ *K*
• accedi al sottoarchivio *P* associato alla chiave *Kh*
• ricerca la chiave *K* nel sottoarchivio *P*

La ricerca all'interno dell'archivio indice può avvenire con qualsiasi metodo di ricerca (sequenziale, binaria, interpolata e così via). In questo tipo di organizzazione, la chiave *Kh* ha la stessa funzione svolta dalla parola riportata nella parte più alta di ogni dizionario. Ricordiamo che, nella precedente figura, le frecce che collegano i sottoarchivi sono state appositamente inserite per suggerire che, nonostante la suddivisione, è possibile eseguire la scansione sequenziale dell'archivio primario utilizzando diverse tecniche. Si può, ad esempio, ricorrere a un puntatore esplicito, per mezzo dello stesso archivio indice, oppure sfruttare la contiguità delle pagine.

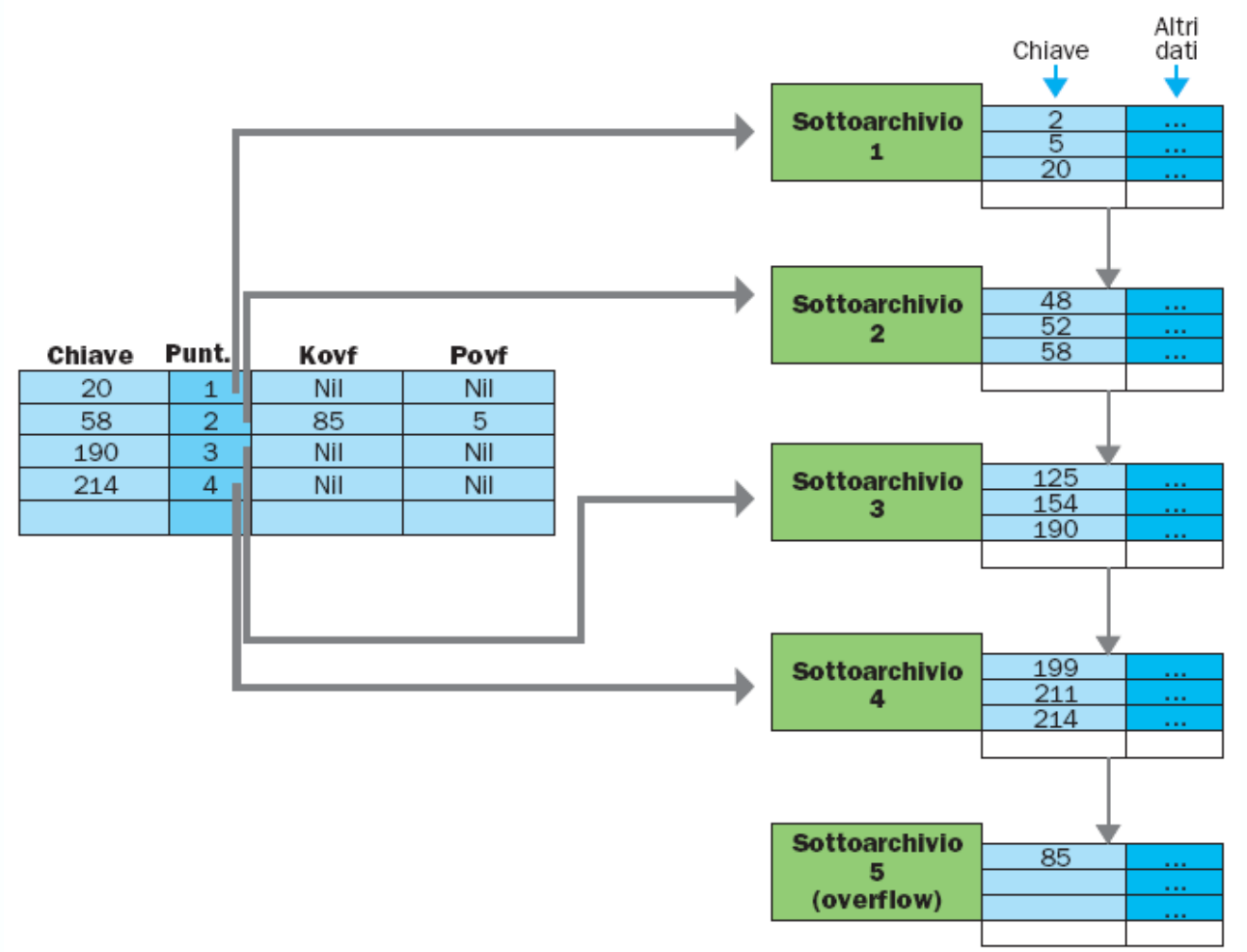
l'aggiornamento

Poiché l'*archivio primario* è pur sempre un *archivio sequenziale*, le operazioni che comportano qualche problema nella loro implementazione sono quelle di *inserimento* e di *cancellazione* di record, le quali richiedono successive riorganizzazioni dell'archivio. Queste operazioni vengono eseguite nello stesso modo previsto per l'organizzazione sequenziale, ossia servendosi di apposite aree di *overflow*. Una tra le tecniche più frequentemente utilizzate prevede che il record dell'archivio indice sia così strutturato: (*Kh*, *P*, *Kovf*, *Povf*)
dove *Kovf* e *Povf* indicano, rispettivamente, la chiave più alta presente nell'area di *overflow* e il puntatore di accesso a tale area. Durante l'esecuzione delle varie operazioni logiche relative a un record di chiave *K*, l'area di *overflow* sarà consultata solo se la pagina *P* è piena e se la chiave *K* risulta compresa tra *Kh* e *Kovf*.
Esempio:



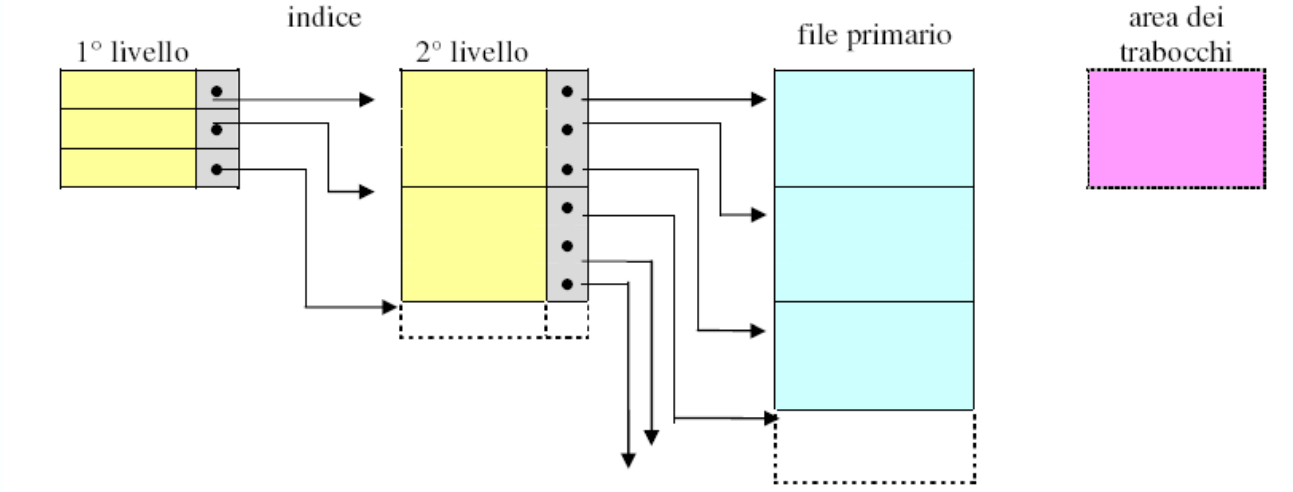
Riprendiamo l'esempio riportato nella precedente lezione e proviamo a inserire il record di chiave 58. L'inserimento verrà effettuato nel sottoarchivio 2 che, essendo pieno, provocherà il trabocco del record di chiave 85.

Dopo l'inserimento la situazione sarà la seguente:



Indici multipli o a più livelli

Nel caso in cui il numero di sottoarchivi diventi rilevante e, di conseguenza, il numero di record presenti nell'indice cominci a divenire considerevole (appesantendo la ricerca), è possibile organizzare l'indice a sua volta come un archivio sequenziale con indice. Si dà così luogo a sottoindici di differente livello, che permettono di ottenere una diminuzione del tempo di scansione dell'indice stesso.In presenza di più livelli di indice, nella fase di ricerca la gerarchia viene utilizzata per poter individuare (partendo da un indice a livello *k*) quale indice a livello*k + 1* debba essere esaminato al fine di selezionare il sottoarchivio all'interno del quale si trova il record cercato.



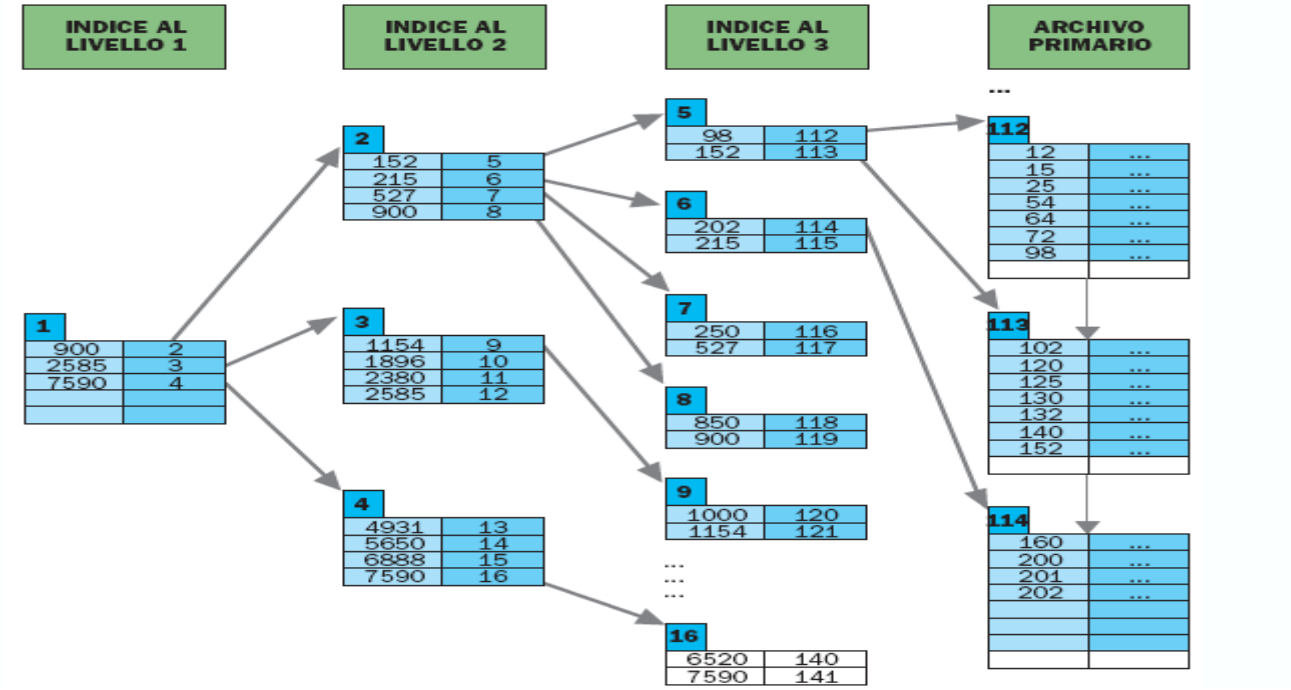
Per comprendere l'uso degli indici a più livelli, si può fare riferimento al reperimento di un'informazione di un volume mediante il suo indice analitico.
Un indice analitico è composto da un insieme di termini accanto ai quali è specificata la pagina del libro all'interno della quale essi sono trattati. I vari termini sono ordinati alfabeticamente e sono raggruppati secondo le lettere dell'alfabeto con la quale iniziano. Se, ad esempio, si vuole ricercare il termine *indice*, si procede nel seguente modo:

1. **si ricerca la lettera *i*;**
2. **nel blocco che racchiude tutti i termini che iniziano con *i*, si ricerca il termine *indice* e si preleva il numero di pagina;**
3. **si apre il libro alla *pagina* ottenuta e si leggono le informazioni.**

Questa struttura è, quindi, un valido esempio di indici multipli: il *primo livello* è rappresentato dalle lettere dell'alfabeto, il secondo livello è costituito da tanti indici quante sono le lettere e ciascun indice è composto da tutti i termini che iniziano con la lettera collegata.
Talvolta, questo tipo di organizzazione viene utilizzato su hard disk, cioè su supporti a disco di tipo *disk-pack*, al fine di ottimizzare i movimenti del pettine. Generalmente, un'organizzazione sequenziale con indice utilizza tre livelli di indice, in particolare:

- **un indice di unità, che consente di determinare su quale unità di memoria secondaria occorre ricercare il record, nel caso in cui l'archivio sia memorizzato su più unità;**
- **per ogni unità, un indice di cilindro che consente di determinare su quale cilindro dell'unità deve avvenire la ricerca;**
- **per ogni cilindro, un indice di traccia che consente di determinare su quale traccia di quel cilindro deve avvenire la ricerca.**

Nel caso di strutture non ordinate, l'indice è più complesso e può essere articolato anche su più livelli. Tuttavia, nonostante il numero di indici e le loro dimensioni ragguardevoli, la gestione non ordinata facilita notevolmente l'operazione di inserimento, poiché nuovi record saranno semplicemente aggiunti in coda all'archivio primario, senza che si verifichino appesantimenti dovuti alla gestione delle aree di *overflow*.



Chiavi Secondarie

In molte applicazioni è necessario individuare velocemente un sottoinsieme di registrazioni in base al valore di uno o più attributi. In tal caso si può definire tale attributo come **Chiave Secondaria**.

Questa chiave non ha il vincolo dell'univocità, come la primaria, e può avere duplicati. Ad esempio, in un archivio anagrafico, potremmo avere come chiave primaria il codice fiscale e come chiavi secondarie il cognome+nome e la città. La gestione della chiave secondaria può essere analoga a quella della chiave primaria, con un Indice o Dizionario per ciascuna diversa chiave (in tal caso nell'esempio precedente si avrebbero tre Indici: codicefiscale, cognome+nome, città) oppure con una organizzazione a parte, specifica. In alcune applicazioni dove occorre avere una efficiente gestione delle chiavi secondarie - si pensi ad esempio ai ***motori di ricerca*** di Internet - queste vengono trattate con specifiche organizzazioni quali quelle a **Liste Multiple** e a **Liste Invertite**.

Organizzazione a Liste Multiple

Per ogni chiave secondaria si costruisce un Dizionario (contenete valore chiave, puntatore inizio lista, lunghezza lista) che punta alla prima registrazione con tale chiave. L'Archivio Dati viene ampliato aggiungendo ad ogni campo che è chiave secondaria un campo puntatore per legare a lista i record con lo stesso valore della chiave secondaria. Se le chiavi secondarie sono più di una, ogni registrazione
apparterrà a più liste. Di qui il termine "*liste multiple*".

Dizionario Città

chiave	inizio lista	lunghezza
FI	3	4
MI	2	3
PG	1	4

Dizionario Reddito

chiave	inizio lista	lunghezza
10	2	5
20	1	5
30	6	3

Archivio dati

	…	Città		…	Reddito		…
1		**PG**	4		**20**	3	
2		**MI**	5		**10**	5	
3		**FI**	6		**20**	4	
4		**PG**	7		**20**	8	
5		**MI**	9		**10**	9	
6		**FI**	8		**30**	7	
7		**PG**	10		**30**	10	
8		**FI**	11		**20**	11	
9		**MI**	0		**10**	0	
10		**PG**	0		**30**	0	
11		**FI**	0		**20**	0	

In inserimento e in cancellazione occorre aggiornare sia i dizionari che le liste dell'archivio dati. In ricerca, per rispondere a interrogazioni composte (AND, OR) tipo

T1 = F1 And F2 And F3
T1 Or T2 Or T3 T2 = F4 And F5
T3 = F6

si accede ai dizionari e si selezionano le liste più corte di ogni termine ***Ti*** dell'interrogazione. Quindi si accede alle registrazioni dell'archivio dati presenti nelle liste e si selezionano quelle che soddisfano l'interrogazione. Le liste debbono essere ordinate in modo che l'elemento successivo abbia un indirizzo più alto. In questo modo, conoscendo gli indirizzi dei prossimi elementi delle liste selezionate, si procede, ad ogni passo, sulla lista che ha l'elemento successivo più vicino.

Organizzazione a Liste Invertite
Per ogni chiave secondaria si costruisce un Dizionario (contenete valore chiave, lunghezza lista, riferimenti a tutte le registrazioni interessate) e l'Archivio Dati resta invariato.

Dizionario Città

chiave	lungh.	riferimenti
FI	4	3, 6, 8, 11
MI	3	2, 5, 9
PG	4	1, 4, 7, 10

Dizionario Reddito

chiave	lungh.	riferimenti
10	3	2, 5, 9
20	5	1, 3, 4, 8, 11
30	3	6, 7, 10

Archivio Dati

	…	Città	…	Reddito	…
1		**PG**		**20**	
2		**MI**		**10**	
3		**FI**		**20**	
4		**PG**		**20**	
5		**MI**		**10**	
6		**FI**		**30**	
7		**PG**		**30**	
8		**FI**		**20**	
9		**MI**		**10**	
10		**PG**		**30**	
11		**FI**		**20**	

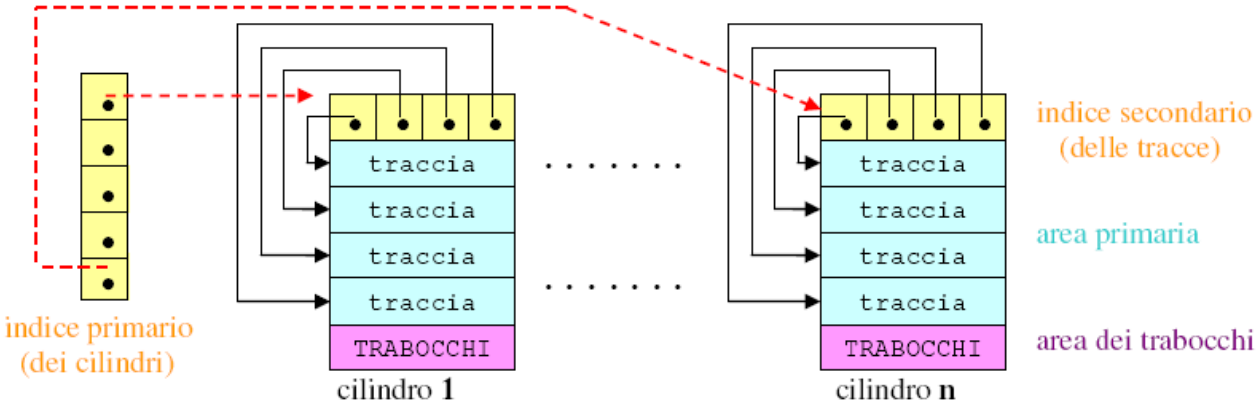
In inserimento e in cancellazione occorre aggiornare i soli riferimenti nei dizionari interessati. Per migliorare la velocità di ricerca, i riferimenti nei dizionari sono ordinati per indirizzi crescenti (come nelle liste multiple).
In ricerca, per rispondere a interrogazioni composte (AND, OR) tipo

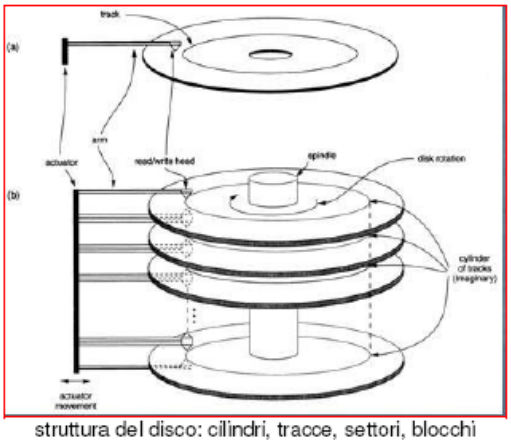
T1 = F1 And F2 And F3
T1 Or T2 Or T3 T2 = F4 And F5
T3 = F6

si accede ai dizionari e si calcola l'***intersezione*** (AND) degli insiemi di riferimento ***Fj***. L'***unione*** (OR) di queste intersezioni costituisce l'insieme degli indirizzi delle registrazioni che soddisfano l'interrogazione. Quindi si accede all'archivio dati e si leggono tali registrazioni. Generalmente questa organizzazione è più efficiente della precedente.

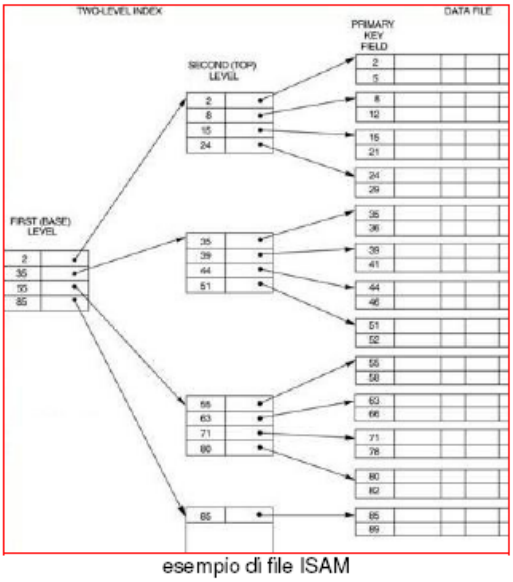
Indici Statici

Una implementazione di un indice a più livelli è l'**ISAM** (*Indexed Sequential Acces Method*) della **IBM**. Tale implementazione usa il disco e la sua struttura (cilindri, tracce) per realizzare i vari livelli dell'indice. Ogni pagina del file primario coincide con una traccia. Le chiavi più alte di ogni traccia e il relativo puntatore (alla traccia) vengono riportate nell'indice di secondo livello, che occupa la prima traccia del cilindro. Le chiavi più alte di ogni cilindro e il relativo puntatore (al cilindro) vengono riportate nel file indice di primo livello (indice dei cilindri).





struttura del disco: cilindri, tracce, settori, blocchi

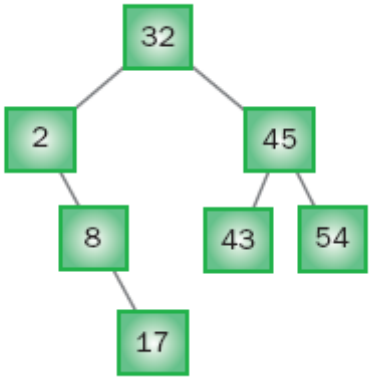


esempio di file ISAM

B-Alberi

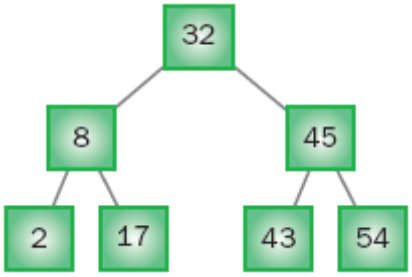
Nel corso della trattazione delle varie organizzazioni abbiamo notato che l'operazione di ricerca diviene sempre più efficiente se i record sono identificati per mezzo delle loro chiavi di accesso e abbiamo visto come l'utilizzo degli indici possa essere di valido aiuto per implementare archivi dinamici.

La gestione degli indici viene resa molto efficiente se si usano gli alberi binari di ricerca per le chiavi, e può essere ottimizzata se si mantiene l'albero bilanciato. La seguente figura mostra un esempio di albero binario di ricerca.



La caratteristica che appare è che partendo dalla chiave di valore 32 a sinistra si va verso chiavi di valore minore e a destra verso chiavi di valore maggiore. Ciò è valido per qualsiasi nodo dell'albero e per qualsiasi chiave che abbia delle altre chiavi sotto di sé.

Quindi gli alberi binari di ricerca hanno una caratteristica ben precisa: possono essere facilmente osservati scorrendoli a partire dalla radice, nodo più in alto.
Per rendere minimo il percorso di ricerca nasce quindi l'esigenza di particolari alberi in cui la profondità si mantenga limitata per tutti i nodi. Una soluzione è bilanciare l'albero, cioè fare sì che tutti i percorsi massimi abbiano la stessa lunghezza.La figura seguente mostra la versione bilanciata del precedente albero binario di ricerca.



Ora, per prelevare il nodo 17 si effettuano tre accessi al file, contro i quattro accessi della versione precedente. Indubbiamente aumenta la complessità delle operazioni di gestione degli alberi. Con l'aggiunta di nuovi nodi, la struttura dell'albero binario aumenta sempre più di dimensioni.
Le caratteristiche di questi alberi implicano che bastano uno o due inserimenti per "sbilanciare" l'albero e costringere alla procedura di bilanciamento, complessa e pesante dal punto di vista del tempo impiegato. Si è dunque cercata un'altra soluzione, di gestione meno impegnativa ma che fornisse prestazioni elevate: i b-alberi.

I b-alberi (***b-tree*** o ***balanced trees*** in inglese) sono strutture di dati introdotte nei primi anni '70 proprio per venire incontro all'esigenza di disporre di modelli implementativi efficienti per la gestione dinamica e ordinata di archivi di dati.
Rappresentano una sorta di estensione degli alberi binari bilanciati di ricerca, poiché sono alberi binari ordinati secondo la chiave e perfettamente bilanciati in altezza. I b-alberi non sono alberi binari e hanno una struttura più complessa di quelli visti in precedenza, ma presentano l'innegabile vantaggio di ridurre enormemente il costo delle operazioni di bilanciamento necessarie per mantenere efficiente la ricerca dopo aver apportato modifiche (inserimenti o cancellazioni di dati) all'albero binario stesso. Le qualità specifiche dei b-alberi possono così sintetizzarsi:

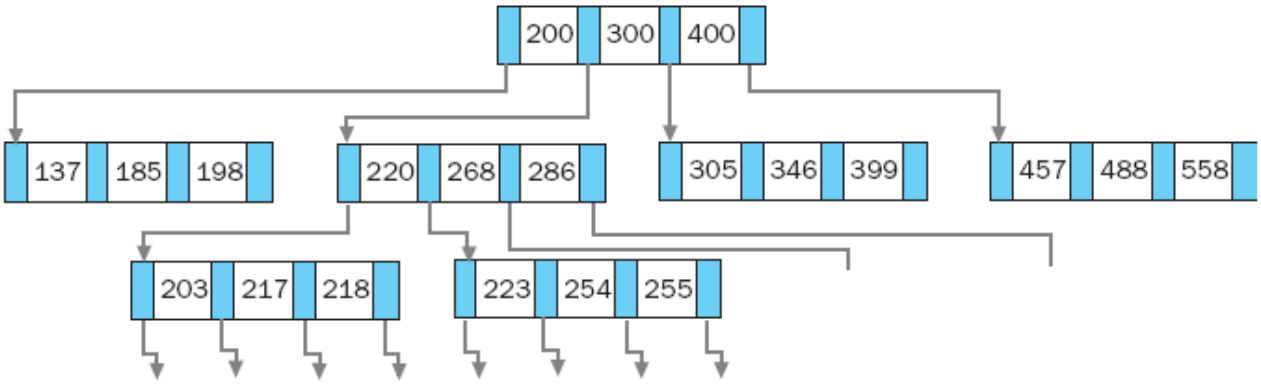
- **offrono ottime prestazioni sia per quanto riguarda le operazioni di ricerca che quelle di aggiornamento poiché entrambe possono avvenire attraverso l'utilizzo di procedure alquanto semplici;**
- **su di essi è anche possibile effettuare elaborazioni di tipo sequenziale dell'archivio primario senza alcuna necessità di sottoporlo a riorganizzazione.**

Diamo, ora, una definizione più formale di b-albero.

Un b-albero di ordine m è un albero che soddisfa le seguenti proprietà:

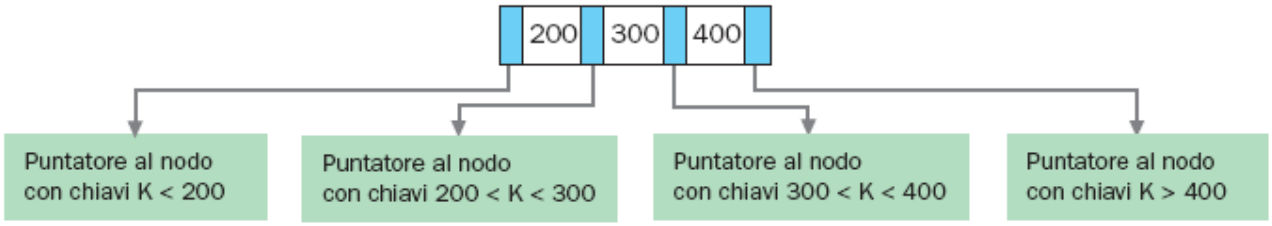
1. **è perfettamente bilanciato in altezza, cioè ha tutte le foglie allo stesso livello;**
2. **il nodo radice ha un numero *d* di figli tale che $1 \leq d \leq 2m + 1$;**
3. **ogni nodo dell'albero, esclusa la radice, ha un numero d di figli tale che: $m \leq d \leq 2m$.**

Un esempio di b-albero di ordine 3 è riportato di seguito. All'interno di ogni nodo di un b-albero è possibile memorizzare più record (chiavi) e da ciascun nodo possono uscire più rami diretti verso altri nodi del b-albero.



La ricerca

La ricerca di un record con una data chiave K in un B-albero viene eseguita con criteri analoghi a quelli visti per gli alberi binari di ricerca; si cerca infatti la chiave K tra le chiavi dei record memorizzati in un dato nodo del B-albero a partire dalla radice.
Nel caso che il nodo attualmente esaminato non contenga il record di chiave K si sceglie quale debba essere il nodo del B-albero da esaminare successivamente in base alla posizione della chiave K rispetto a quelle già memorizzate nel nodo; il criterio per stabilire se una ricerca termina senza successo è analogo a quello per gli alberi binari di ricerca.
Per comprendere meglio il criterio di ricerca in un B-albero si può fare riferimento alla situazione descritta nella seguente figura.



La figura illustra che una qualunque chiave che non appartenga alla radice deve necessariamente trovarsi in una delle parti nelle quali la linea è suddivisa dalle chiavi che si trovano in essa. A ciascuna suddivisione della linea è associato il puntatore che si riferisce al sottoalbero dell'albero di ordine 3 nel quale sono memorizzati tutti e soli i record che hanno la chiave nell'intervallo corrispondente.
Per continuare la ricerca del record con chiave K si deve pertanto accedere al sottoalbero corrispondente all'intervallo nel quale K si trova; il procedimento prosegue finché non si ritrova il record con chiave K (ricerca con successo) oppure è più possibile procedere all'interno dell'albero di ordine n (ricerca senza successo). Più in dettaglio, il nodo di un B-albero ha la seguente struttura:

dove:

- n rappresenta il numero di voci presenti nel nodo;
- il puntatore P0 al primo figlio di sinistra consente di accedere al sottoalbero che contiene tutte le chiavi aventi valore minore della più piccola (ossia di K1);
- i puntatori intermedi Pi permettono l'accesso al sottoalbero che contiene tutte le chiavi aventi valori compresi tra Ki e Ki+1;

- il puntatore Pm all'ultimo figlio di destra consente di accedere al sottoalbero che contiene tutte le chiavi aventi valore maggiore della chiave più grande (ossia di Km).

Alla luce di queste prime definizioni, possiamo concludere che un B-albero gode delle seguenti proprietà:
• è perfettamente bilanciato in altezza, cioè ha tutte le foglie allo stesso livello;
• ogni nodo, esclusa la radice, deve contenere almeno *m* chiavi;
• ogni nodo, inclusa la radice, deve contenere al massimo 2*m chiavi*;
• ogni nodo ha le chiavi disposte in ordine crescente;
• un nodo può essere una foglia, ossia una pagina terminale.

L'inserimento nel b-albero

L'operazione di inserimento, ma anche quella di cancellazione, rappresentano le operazioni più complesse e delicate, poiché devono garantire che la struttura rimanga sempre bilanciata. Nel caso in cui si vada a inserire all'interno di un nodo saturo (cioè di un nodo che contiene già 2m voci) occorrerà allocarne uno nuovo. Riferiamoci alla successiva figura e supponiamo di dover inserire la chiave 9. Tale chiave dovrebbe essere inserita all'interno di un nodo saturo.
Poiché ciò non è possibile occorrerà sdoppiare il nodo (splitting). Le chiavi saranno divise in parti uguali e l'elemento centrale, cioè 8, viene trasferito nel nodo di livello superiore se c'è posto, altrimenti si ripete il procedimento.