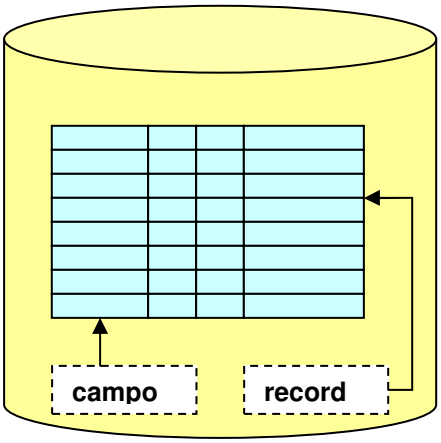
Archivi

prof. Claudio Maccherani - Perugia - 2009 .

Generalità sugli ARCHIVI



Un **archivio** di dati è un file (memorizzato su disco) 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 una informazione elementare.
Un record fisico è l'insieme dei dati letti o scritti ad ogni lettura o scrittura fisica su disco (un record fisico corrisponde ad un **blocco**, incrocio traccia-settore, ed ha dimensione fissa, generalmente 521, 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.

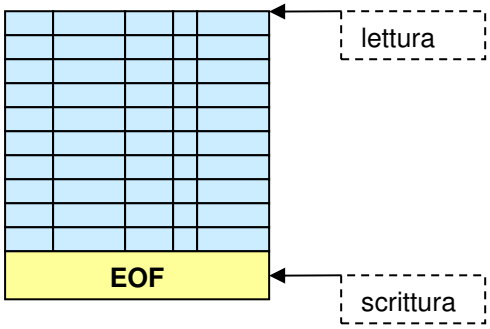
L'archivio, prima dell'utilizzazione, deve essere "*aperto*" con una opportuna istruzione di apertura e dopo l'utilizzazione deve essere "*chiuso*".

Esistono diverse **organizzazioni** degli archivi - sequenziale, casuale, a indici, sequenziale con indice - ciascuna delle quali caratterizzata dal metodo di **accesso** ai record.

Parlando di "archivi dati" è più appropriato parlare di ***archivi tradizionali***, in quanto ormai essi sono stati sostituiti dalla loro evoluzione, i ***database***.

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.



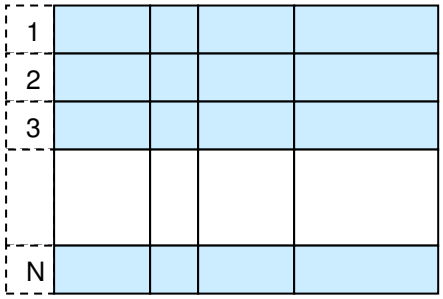
L'archivio può essere aperto in *input/read* (per lettura, partendo dall'inizio), in *output/write* (per scrittura partendo dall'inzio: se l'archivio non esiste lo cera, altrimenti lo cancella), in *append* (per scrittura partendo dalla fine).

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).

Organizzazione CASUALE (2)

In un archivio **casuale** (o **random**, o **relativo**, o ad **accesso diretto**) è possibile accedere ai record, per memorizzarli e/o per leggerli, direttamente, semplicemente specificando quale 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 utilizzarno sono del tipo: *lettura record **n***, *scrittura record **n*** (dove "***n***" è il numero del record da leggere e/o scrivere).

In questo caso l'archivio può anche essere visto come un "*vettore di record*" in quanto si accede al singolo record specificando la sua posizione.

Con questa organizzazione i record debbono essere a lunghezza fissa (mentre, con gli archivi sequenziali, si possono avere anche record a lunghezza variabile).

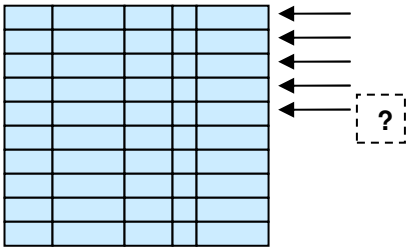
Per variare uno o più campi del record basta leggere il record, modificare i valori e riscriverlo.
La cancellazione è logica, basta memorizzare sopra il record da cancellare un record vuoto.

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

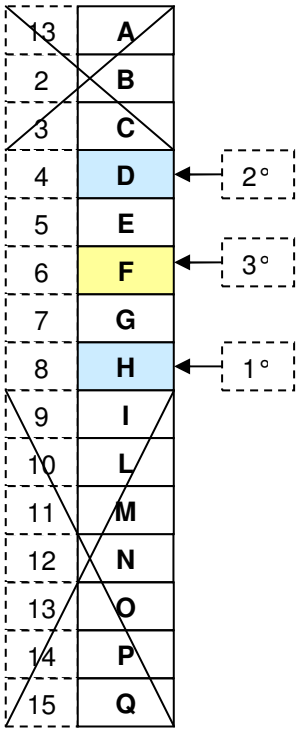
L'organizzazione casuale è ottima quando si conosce a priori la posizione del record da leggere e/o da scrivere, quando cioè la ***chiave di accesso*** al record è un numero intero da 1 a N.
Le cose, però, si complicano quando la chiave di accesso al record non è la sua posizione. Ad esempio in un archivio 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. Tale ricerca funziona così:

- si accede alla metà dell'archivio e si confronta la chiave cercata con quella trovata;
- se la chiave trovata è quella cercata la ricerca è finita;
- se la chiave trovata è maggiore di quella cercata allora ciò che si cerca, se esiste, sarà nella parte superiore dell'archivio e allora si "elimina logicamente" la parte inferiore dell'archivio e si riapplica la ricerca binaria sulla parte superiore (da 1 a N/2);
- se la chiave trovata è minore di quella cercata allora ciò che si cerca, se esiste, sarà nella parte inferiore dell'archivio e allora si "elimina logicamente" la parte superiore dell'archivio e si riapplica la ricerca binaria sulla parte inferiore (da N/2 a N);
- la ricerca termina quando si trova ciò che si cerca o quando l'archivio è ridotto a 0 record (nel qual caso si può affermare che ciò che si cerca non esiste).



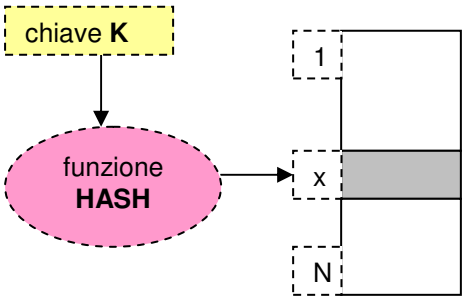
Con questo metodo il numero massimo **x** di accessi è $\log_2 N$, cioè $x \leq 2^N$.

Più N è grande, più è veloce questa ricerca (paragonata alla ricerca sequenziale); l'unico scotto da pagare è il fatto che l'archivio deve essere mantenuto ordinato sulla chiave di ricerca.

N.record	ricerca **sequenziale**	ricerca **binaria**
10	5	4
100	50	7
1.000	500	10
10.000	5.000	14
100.000	50.000	17
1.000.000	500.000	20

Organizzazione con INDICE (3)

In un archivio a **indice** (o **index** o **HASH** o a **trasformazione della chiave**) si può accedere 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(Hi) ≠ 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 ∈ [A,B,C,…,Z] e R ∈ [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) moulo 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**. Per un approfondimento sulla gestione delle collisioni si veda l'***Appendice A***.

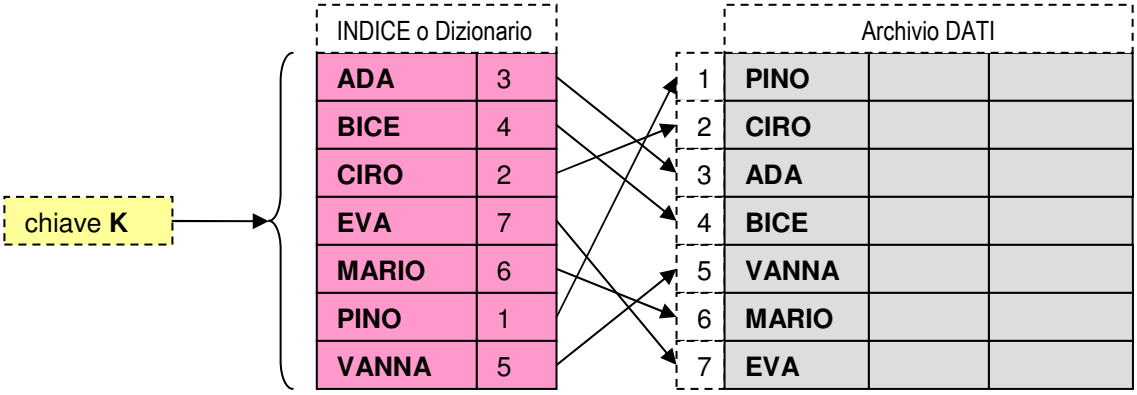
Con questa organizzazione si possono usare istruzioni di accesso diretto tipo *leggi record con chiave K*, e *memorizza record con chiave K*, ma non si possono reperire i record sequenzialmente in ordine di chiave, dalla più piccola alla più grande.

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".

Organizzazione SEQUENZIALE con INDICE (4)

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**.



L'accesso avviene tramite il Dizionario dove la chiave viene velocemente reperita (ad esempio possiamo immaginare con una ricerca binaria) unitamente al puntatore che indica su quale record dell'archivio si trova la registrazione ad essa associata. Il Dizionario viene mantenuto logicamente ordinato quando si registra un nuovo record e quando si cancella un record, nonché quando si modifica il valore di una chiave.

Le possibili istruzioni che si possono utilizzare in un archivio sequenziale con indice sono:

- *memorizza il record con chiave K*
- *cancella il record con chiave K*
- *leggi il record con chiave K*
- *leggi il record con chiave ≥prefisso*
- *leggi il record associato alla prima chiave* *First* *Key*
- *leggi il record associato alla prossima chiave* *Next* *Key*
- *leggi il record associato all'ultima chiave* *Last* *Key*
- *leggi il record associato alla chiave precedente* *Previous* *Key*

Quindi è possibile sia l'accesso diretto, specificando il valore della chiave, che l'accesso sequenziale.
Con questa organizzazione non si ha sia l'**EOF** (*End Of File*) che il **BOF** (*Bottom Of File*) che indicano, rispettivamente, quando si è superata la fine o l'inizio del file..

Esistono vari metodi di gestione del file Indice (o Dizionario), più o meno potenti, più o meno flessibili, più o meno veloci. Sostanzialmente possiamo avere un **Indice Statico** - ad un livello o a più livelli, come l'**ISAM** (*Indexed Sequential Acces Method*) della IBM - o un **Indice Dinamico** basato su strutture **B-Tree**, estensioni dell'ABR (*Albero Binario di Ricerca*), quali B-Albero, B*-Albero, B+_Albero. Per un approfondimento sulla gestione degli indici si veda l'***Appendice B***.

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: codice fiscale, 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 <u>Liste Multiple</u>

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 Or T2 Or T3

T1 = F1 And F2 And F3
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 <u>Liste Invertite</u>

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 Or T2 Or T3

T1 = F1 And F2 And F3
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.

Altre organizzazioni

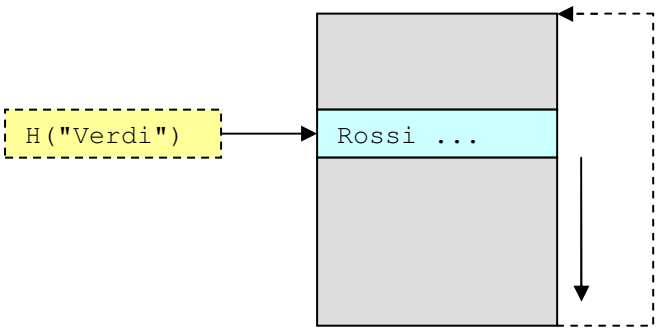
Per la gestione delle chiavi secondarie esistono altri tipi di organizzazione che attingono ai metodi sopra descritti, integrandoli in misura diversa.

Appendice A - Gestione delle COLLISIONI

Una **funzione HASH** (**H**) trasforma una chiave alfanumerica in un numero record. Tali funzioni, nella quasi totalità dei casi, danno adito a "collisioni": H(Ki) = H(Kj) con Ki ≠ Kj (cioè la funzione applicata a due chiavi diverse restituisce lo stesso numero record).
Tali collisioni, praticamente *inevitabili*, debbono in qualche modo essere gestite.

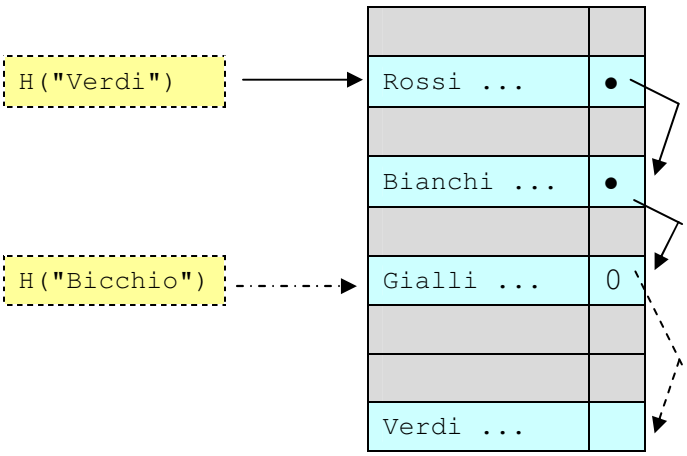
A - **Scansione lineare**

In caso di collisione si scandiscono in sequenza i record successivi fino a trovarne uno libero dove memorizzare la collisione o a tornare al record di partenza (considerando l'archivio come *circolare*), nel qual caso l'archivio è pieno.
In questo modo le collisioni vanno ad occupare record riservati ad altre chiavi, aumentando esponenzialmente la possibilità di successive collisioni.



In lettura, se un record è occupato da una chiave diversa da quella cercata, occorre leggere in sequenza tutti i record (in maniera *circolare*) fino a che non si trova la chiave richiesta o fino a tornare al record di partenza, se la chiave non esiste. Per decidere la non esistenza, cioè, occorre leggere tutto l'archivio.

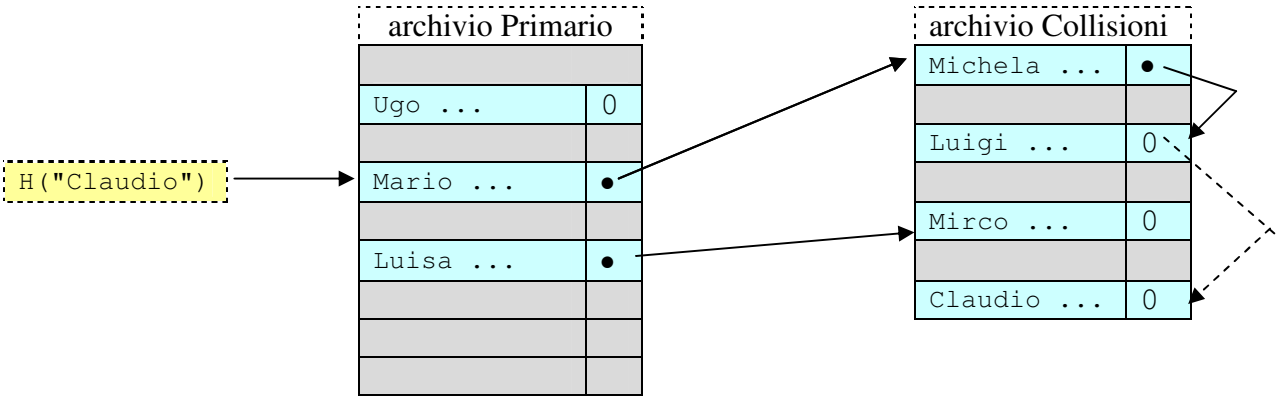
B – **Catene confluenti**



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. In un'unica lista, cioè, 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

C – **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.

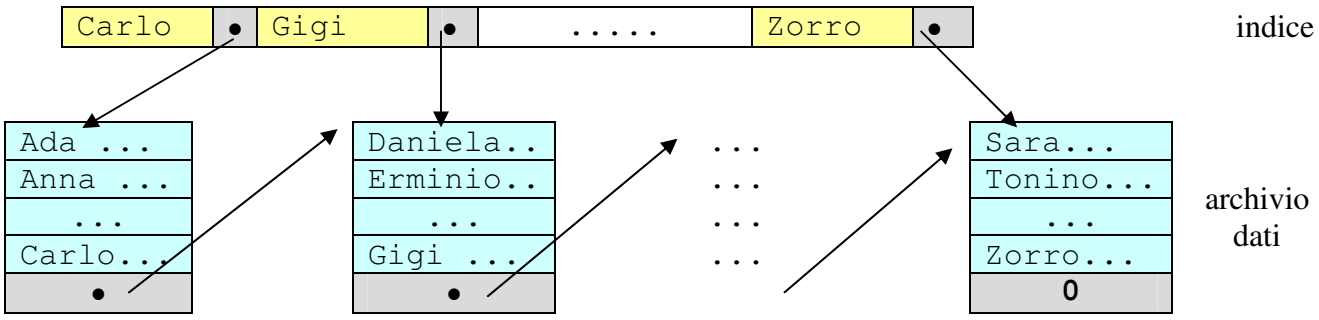


Appendice B - Gestione INDICE (o DIZIONARIO)

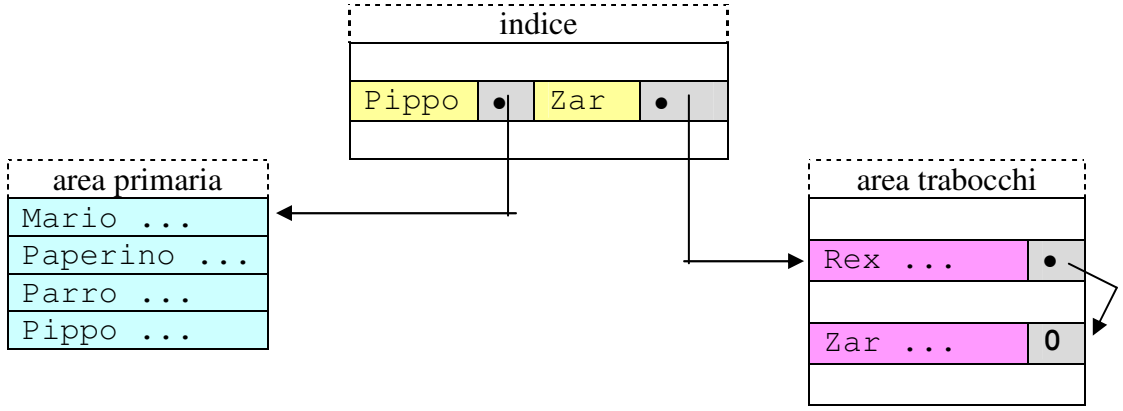
Esistono varie tecniche di gestione del file Indice (o Dizionario) degli archivi Sequenziali con Indice. Storicamente sono stati utilizzati indici statici ad uno o più livelli. Attualmente si utilizzano indici dinamici basati su strutture B-Tree (albero bilanciato), estensioni dell'ABR (Albero Binario di Ricerca). E questo vale anche per quanto riguarda l'indicizzazione delle tabelle nei sistemi di gestione database.

Indici STATICI

Ogni pagina dell'archivio dati contiene un certo numero di registrazioni ordinate sulla chiave. La chiave più alta di ogni pagina è riportata sul file indice che è costituito da record di tipo [Kp, P], dove Kp è la chiave più alta della pagina P del file dati. Le pagine del file dati sono legate a lista. Il file indice è ordinato secondo la chiave. La ricerca nel file indice e nel file dati è fatta mediante *ricerca binaria*.

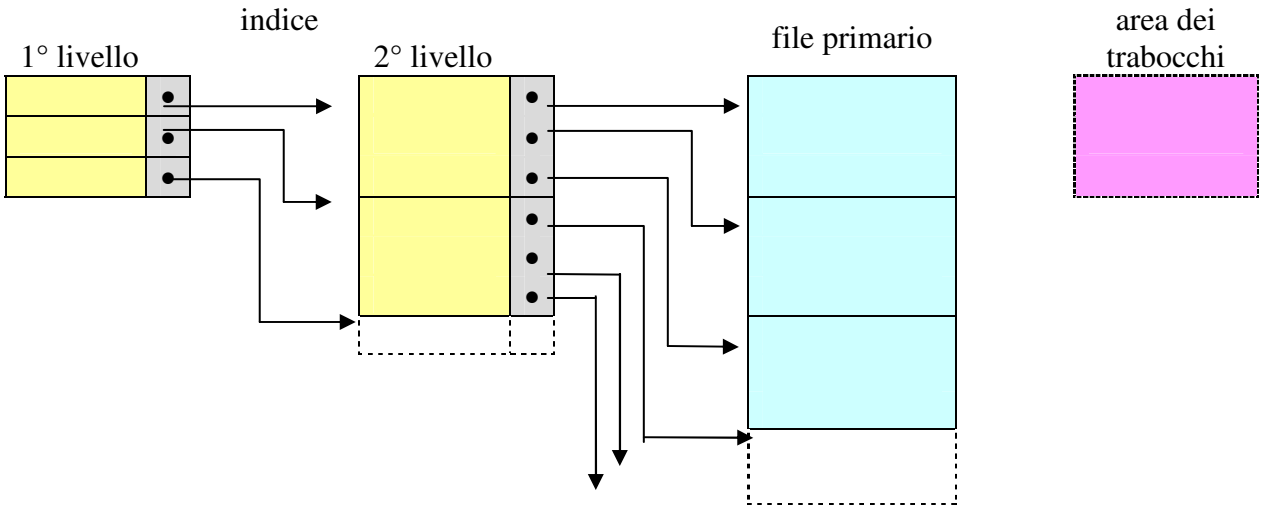


In inserimento e in cancellazione potrebbe essere necessaria una ridistribuzione delle registrazioni tra le pagine del file dati (ad esempio se si deve inserire una registrazione in una pagina piena) e la conseguente modifica del file indice. Per evitare o limitare o ritardare questa riorganizzazione degli archivi dati e indice (in seguito a inserimenti e cancellazioni) si può usare la ***lista di overflow*** (o ***lista dei trabocchi***, che dir si voglia). In tal caso il record del file indice è del tipo [Kp, P, Kow, Pow] dove Kp è la chiave più alta della pagina P del file dati, Kow è la chiave più alta della lista di overflow e Pow è il puntatore di accesso alla lista di overflow.

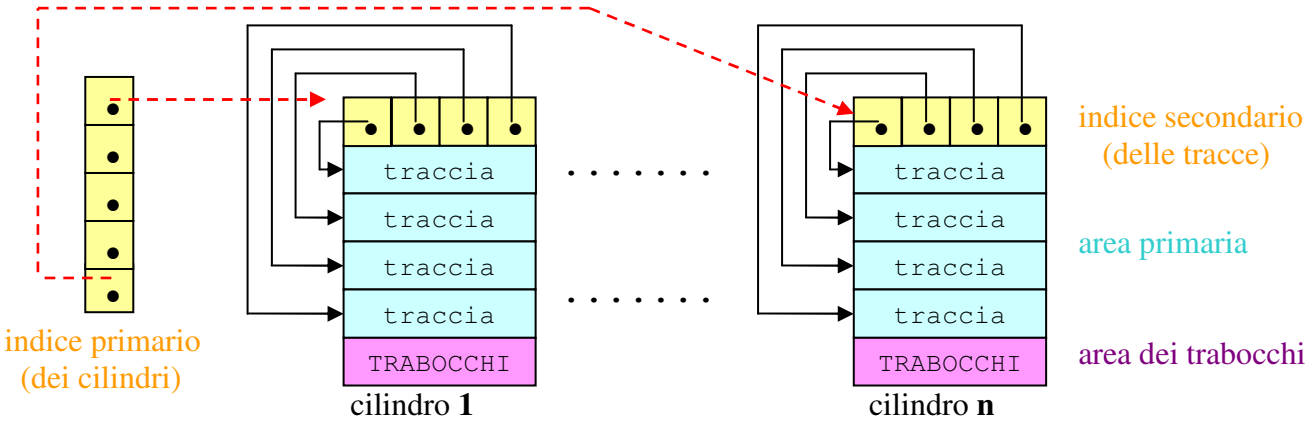


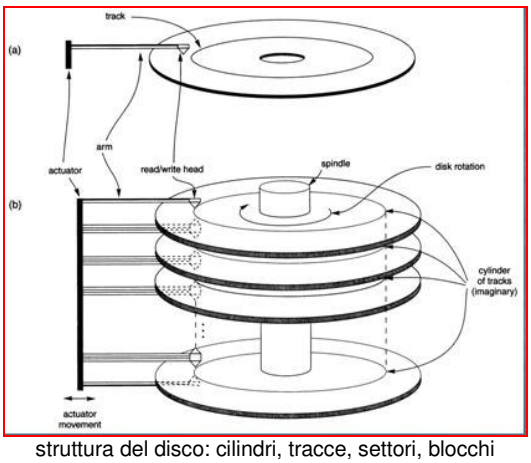
In inserimento, se la pagina è piena, si utilizza la pagina di overflow; se la chiave da inserire è maggiore di Kp viene aggiornata la lista di overflow, altrimenti la Kp passa nella lista di overflow, la chiave viene inserita nella pagina e il record del file indice viene modificato opportunamente.
La cancellazione può essere logica (il record viene "marcato" e il recupero dei record marcati viene rinviato all'eventuale riorganizzazione del file primario) o fisica, con conseguente aggiornamento della lista di overflow, del record indice e ricompattamento della pagina.
Generalmente l'area dei trabocchi consiste in una pagina per ogni **n** pagine del file primario.

File indice a più livelli. Se l'indice viene a sua volta organizzato come file sequenziale con indice si ottiene un file indice a due livelli.

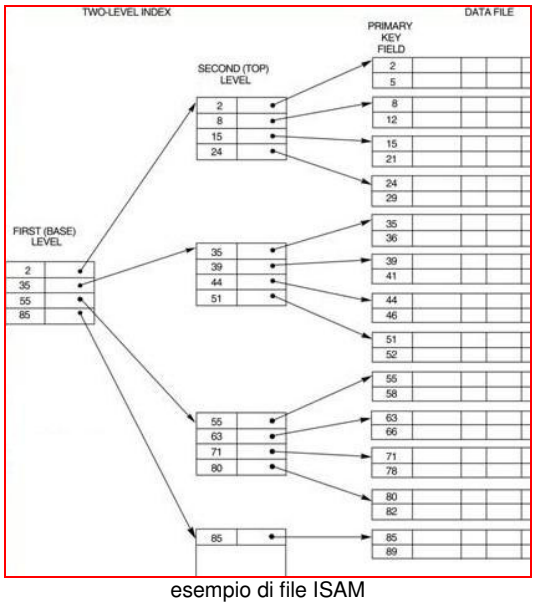


Una implementazione di un indice a più livelli è l'**ISAM** (*Indexed Sequential Acces Method*) della **IBM**. Tale implementazione usa il disco e la sua struttura (cilindri, tracce) per realizzare i vari livelli dell'indice. Ogni pagina del file primario coincide con una traccia. Le chiavi più alte 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

Indici DINAMICI

Gli indici dinamici sono realizzati con implementazioni basate su generalizzazioni dell'**A.B.R.** (*Albero Binario di Ricerca*).

B-Albero (*B albero, Albero Bilanciato*) o ***B-Tree*** (*Balanced Tree*)

Un B-Albero è una generalizzazione dell'ABR, è ordinato secondo la chiave ed è bilanciato. Le operazioni di ricerca e di modifica sono logaritmiche, le operazioni di inserimento e cancellazione garantiscono che il B-Albero resti sempre bilanciato.
Un B-Albero di ordine **M** (**M** ≥ 3) è un albero con le seguenti proprietà:

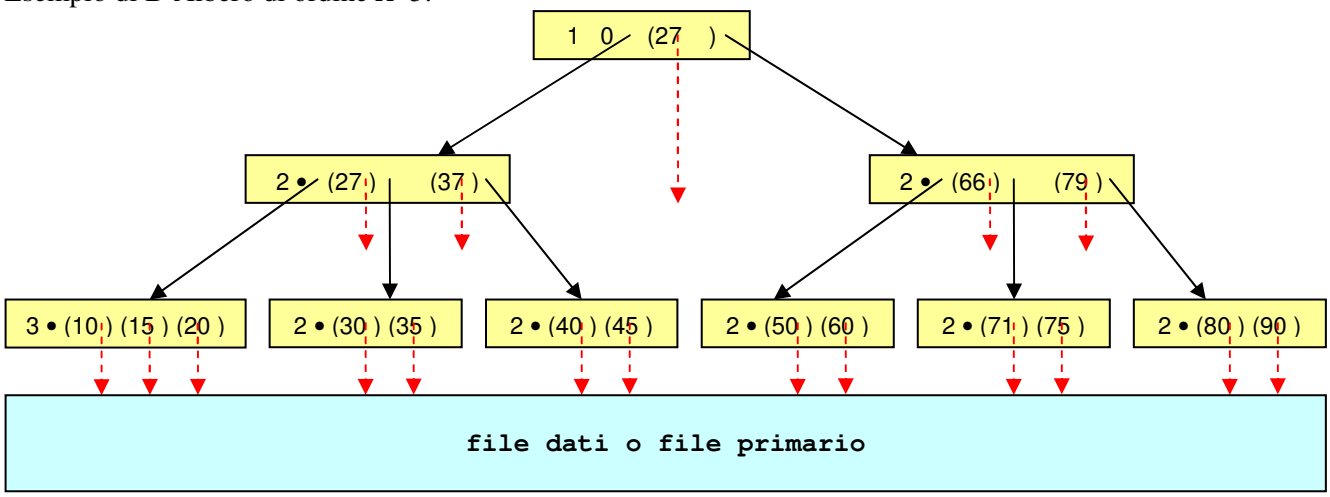
a) ogni nodo ha al massimo **M**-1 elementi
b) ogni nodo, tranne la radice, ha almeno [**M**/2] – 1 elementi
c) ogni nodo, tranne la radice, ha **m** + 1 figli, se **m** sono i suoi elementi
d) tutti i nodi terminali - foglie - sono allo stesso livello (albero bilanciato)
e) ogni nodo, che coincide con un pagina o blocco o record fisico, ha la seguente struttura:

$$J,\ P,\ P_0,\ (K_1,\rho_1),\ P_1,\ (K_2,\rho_2),\ P_2,\ \ldots\ ,\ (K_m,\rho_m),\ P_m$$

J = numero di elementi (chiavi) del nodo
m = numero di elementi (chiavi) del nodo
P = puntatore al padre (puntatore interno)
K_i = chiavi (elementi) del nodo
ρ_i = puntatori ai record associati alle chiavi K_i (puntano esterni, al file dati)
P_i = puntatori ai figli (puntatori interni)
Le chiavi sono ordinate: $K_1 < K_2 < \ldots < K_m$

f) per ogni nodo non terminale valgono le seguenti proprietà:
f1) B_0 è un B-sottoalbero (puntato da P_0) che contiene chiavi minori di K_1
f2) B_i ($1 \le i \le m$) è un B-sottoalbero (puntato da P_i) che contiene chiavi comprese tra K_i e K_{i+1}
f3) B_m è un B-sottoalbero (puntato da P_m) che contiene chiavi maggiori di K_m

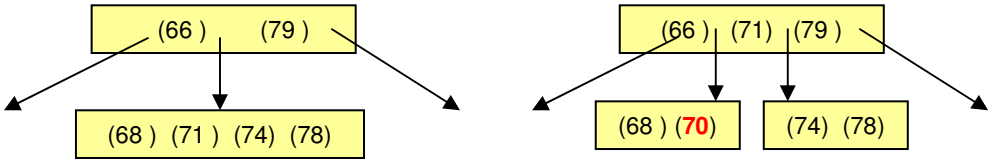
Esempio di B-Albero di ordine **M**=5:



Visitando il B-Albero in un ordine che è una naturale estensione di quello **simmetrico** si ritrovano le chiavi in ordine crescente.
Il numero massimo di accessi al B-Albero in ricerca è **L**, dove **L** è il numero di livelli del B-Albero.
Se **n** è il numero delle chiavi si ha: $\log_M(n+1) \le \mathbf{L} \le \log_{[M/2]}((n+1)/2)$

Esempio: **n** = 2.000.000, **M** = 200 ⇒ 2,74 ≤ **L** ≤ 4 (3 o 4 livelli)
n = 1.048.576, **M** = 32 ⇒ 4 ≤ **L** ≤ 4,75 (4 o 5 livelli)

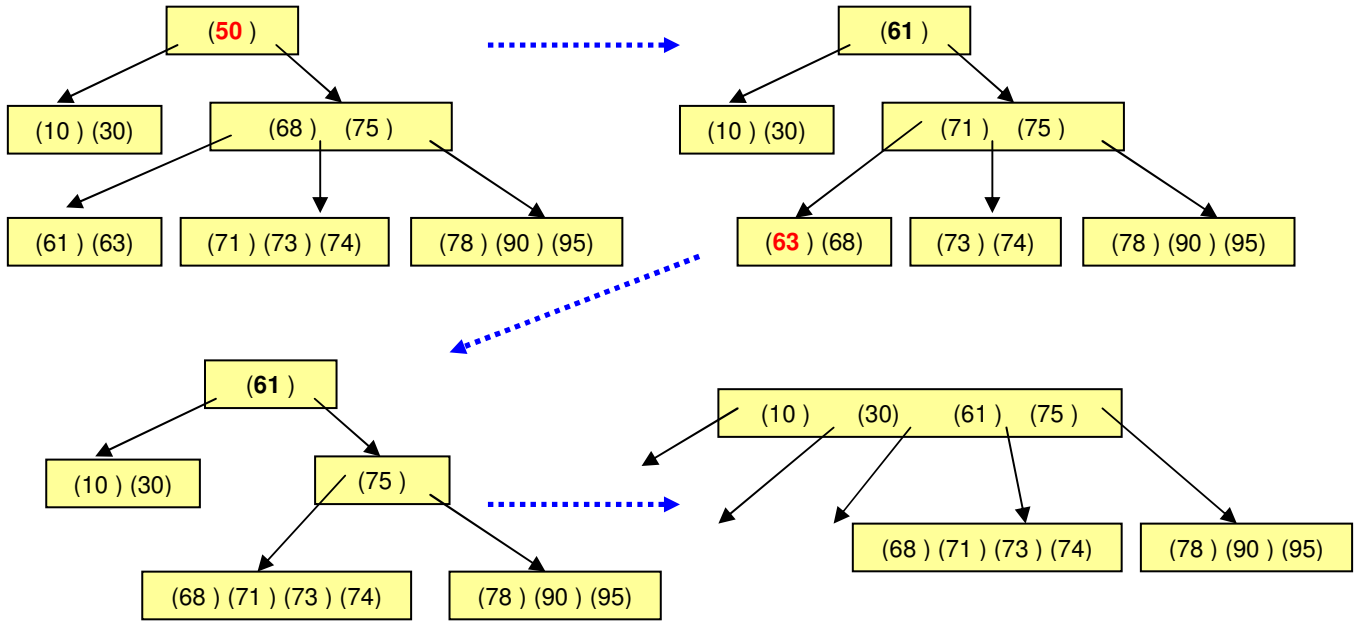
Per l'inserimento di una chiave nel B-Albero si applica l'algoritmo di ricerca fino a aggiungere un nodo terminale, nel quale si inserisce l'elemento. Se però tale nodo ha già il massimo di elementi (M-1) occorre sdoppiarlo e ribilanciare l'albero, come mostrato nel seguente esempio di inserimento della chiave **70**:



Tale sdoppiamento può propagarsi, al limite, fino alla radice, nel qual caso l'albero aumenta di un livello. Una variante dell'algoritmo di inserimento consiste nel tentare, prima dello sdoppiamento, una operazione di ridistribuzione delle chiavi fra nodi fratelli adiacenti (se un nodo è saturo, ma un nodo adiacente non è saturo si può ridistribuire e ribilanciare l'albero, senza bisogno di sdoppiare).

Per quanto riguarda la cancellazione, si possono cancellare solamente le chiavi che si trovano su un nodo terminale. Se la chiave è su un nodo non terminale occorre, prima della cancellazione, spostarla su un nodo terminale. Se il nodo terminale ha più di [M/2]-1 chiavi si cancella solamente, altrimenti occorre anche una ridistribuzione delle chiavi (bilanciamento) oppure una operazione di concatenazione con un nodo adiacente (fratello) con [M/2] elementi.

Esempio di cancellazione della chiave **50**, prima, e poi della chiave **63** (con concatenazione):



La procedura di cancellazione può propagarsi fino alla radice, nel qual caso l'albero diminuisce di un livello.

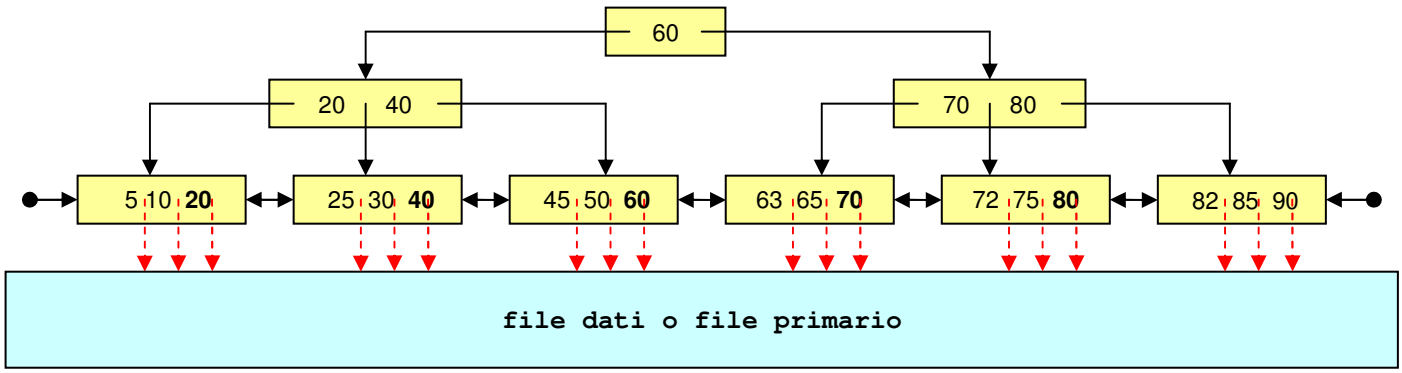
B-Albero* (*B "star" albero*)

Un B*-Albero è ottenuto dal B-Albero dal quale differisce per le seguenti caratteristiche:

- i nodi non terminali hanno solo chiavi e puntatori ai figli
- i nodi terminali hanno chiavi e puntatori ai record dell'archivio dati
- nei nodi terminali P_0 punta al nodo adiacente minore e Pm punta al nodo adiacente maggiore.

Per la ricerca occorre sempre leggere L livelli, ma, grazie ai puntatori orizzontali che collegano i nodi terminali, è più veloce la scansione sequenziale.

Esempio di B*-Albero:

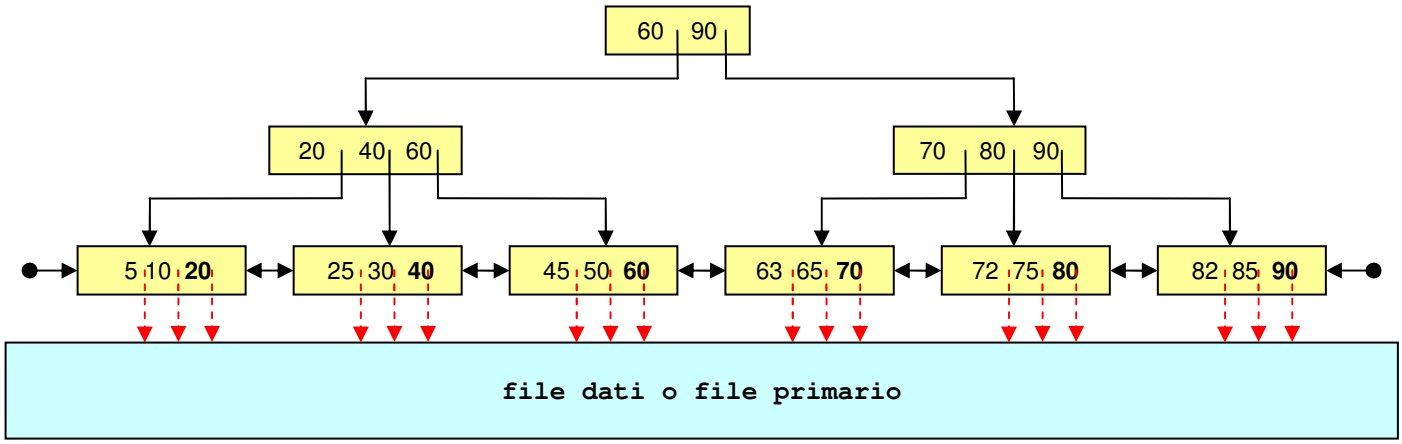


B^+-Albero (*B "plus" albero*)

Un B^+-Albero è ottenuto dal B*-Albero eliminando il puntatore P_0 dei nodi non terminali e ripetendo in ogni nodo non terminale la chiave più alta presente nel nodo figlio.

È l'organizzazione più utilizzata. È usata dalla **IBM**, con il nome di **VSAM**, per il suo database **DB2**.

Esempio di B^+-Albero:



(*or che bravo sono stato, posso fare anche il bucato*)