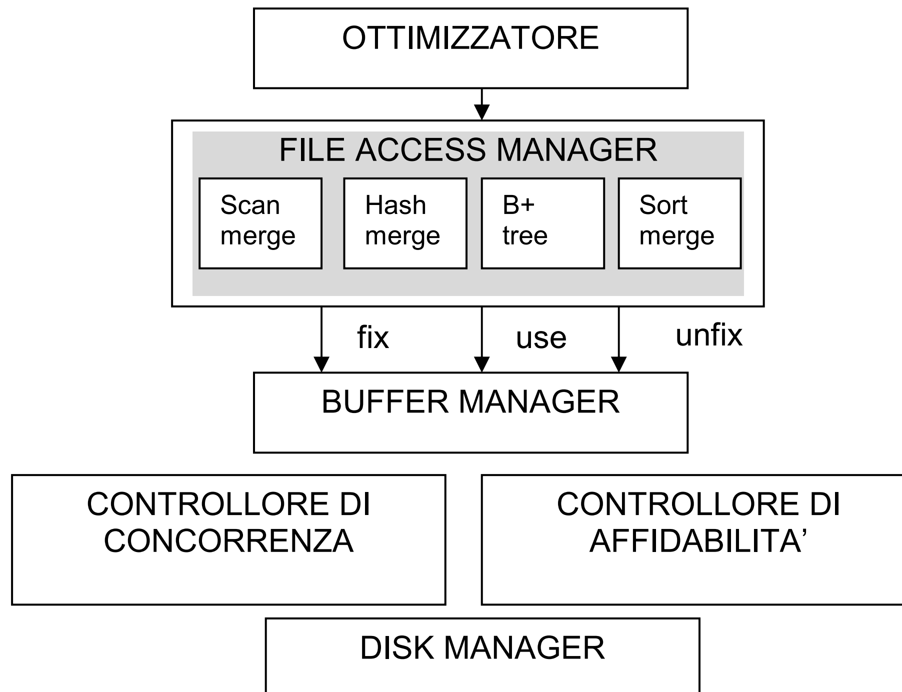
ARCHITETTURA DI UN B.D.M.S.

Parte IV – I Metodi di Accesso ai Dati

Michele de Nittis

Generalità

Si consideri nuovamente la struttura astratta di un DBMS:



Il compito del servizio denominato Gestore dei Metodi di Accesso (*File Access Manager- FAM*) è quello di tradurre i piani di esecuzione elaborati dei livelli superiori in opportune sequenze di accesso alle pagine di memoria.

In generale le modalità di accesso alle pagine di dati si diversificano in funzione del tipo di operazione che deve essere compiuta.

Dunque, un piano d'esecuzione prodotto dai livelli superiori prevede un certo numero di operazioni che richiedono l'accesso ai dati, ciascuna secondo il metodo di accesso più opportuno in termini di efficienza.

Ciascuno dei **metodi di accesso,** che possiamo considerare come funzioni **primitive,** espletate da appositi moduli software disponibili al FAM, che hanno il compito di **localizzare** le pagine all'interno dei file ed i dati all'interno delle pagine, fa riferimento ad una ben definita **organizzazione dei dati** (nelle pagine).

Indipendentemente dal tipo di organizzazione adottata, una generica pagina di memoria contiene le seguenti informazioni:

1. un **header** contenente informazioni di controllo e di gestione utili per il file system;
2. un **header** ed un **trailer** contenenti informazioni di controllo utili per uno specifico metodo di accesso;

3. un **dizionario di pagina**, contenente i **puntatori** ai dati utili della pagina;
4. la **parte utile**;
5. bit di parità o CRC;

Generalmente la parte utile ed il dizionario di pagina crescono come stack contrapposti.

Le tuple (o record) all'interno di una pagina possono essere a lunghezza fissa, più semplici da gestire, o lunghezza variabile (più efficienti dal punto di vista dell'impiego di memoria).

Nel seguito, nell'illustrare le diverse possibili organizzazioni dei dati nelle pagine, si farà riferimento alle seguenti operazioni di accesso ai dati:

- Inserimento/modifica;
- Cancellazione;
- Scansione;
- Accesso ad un particolare record (selezione su uguaglianza) o a un suo campo;
- Accesso ad un range di record;

Esistono due tipologie di organizzazione dei dati all'interno delle pagine:

- Le Strutture Sequenziali;
  - Heap o Entry Sequenced;
  - Array;
  - Sorted;
  - Hash (accesso calcolato).
- Le Strutture Ausiliarie, tra queste le più comuni sono:
  - Indici ISAM;
  - Indici ad Albero;
  - Indici hash.

Le strutture sequenziali

Sono caratterizzate da una disposizione sequenziale dei record (tuple) nelle pagine di memoria.

Struttura HEAP o Entry Sequenced

Questa struttura è molto semplice poiché non vi è alcun ordine delle tuple all'interno delle pagine del file.

Il caricamento iniziale del file di dati, le selezioni e gli accessi ai singoli record avvengono per scansione del file mentre gli inserimenti avvengono in coda. Le operazioni di cancellazione o di modifica della struttura dei record implicano la riorganizzazione dei record nel file oppure la presenza di blocchi di memoria inutilizzati.

Ricapitolando: la struttura HEAP è molto semplice e caratterizzata da operazioni di inserimento molto veloci. E' particolarmente conveniente qualora siano frequenti operazioni di lettura che interessino tutti i record del file (scansione) ma non è indicata per frequenti accessi in selezione, modifica ed inserimento di record.

Struttura ad Array (accesso diretto)

Questa struttura organizzativa, come dice il nome, prevede che l'accesso ai record avvenga direttamente attraverso un indice posizionale ed è praticabile solo qualora i record siano di dimensione fissa.

Il caricamento iniziale dei dati avviene attraverso il continuo incremento dell'indice, come in una scansione. Le operazioni di cancellazione non comportano la riorganizzazione dei record del file perché gli eventuali spazi vuoti possono essere individuati mediante l'indice. Gli inserimenti avvengono negli spazi vuoti o in coda. Le operazioni di selezione di un record o di un insieme di record avvengono per scansione.

Ricapitolando: la struttura ad ARRAY è indicata per l'accesso diretto ad un record e per le operazioni di inserimento, modifica e cancellazione. Non presenta particolari vantaggi sulle altre strutture per quanto concerne le operazioni di selezione e di scansione del file.

Struttura Sequenziale Ordinata (Sorted)

Questa struttura organizzativa prevede che la posizione di un record all'interno di un file sia determinata dal valore di uno o più campi. In altre parole, i record all'interno del file sono ordinati sul valore assunto da uno o più campi denominati **chiave**.

Questa organizzazione consente un **accesso diretto** al dato di tipo **associativo**, ovvero in base al valore assunto dalla chiave.

Il caricamento iniziale del file avviene attraverso la sua scansione completa, mentre le operazioni di cancellazione e d'inserimento comportano la riorganizzazione dei record all'interno del file e, pertanto, sono poco efficienti. Anche le operazioni di modifica della struttura dei record implicano la loro riorganizzazione all'interno del file. Le operazioni di ricerca e di selezione sono veloci ed efficienti se avvengono sui campi che costituiscono la chiave, altrimenti comportano la scansione del file. E' possibile evitare che le operazioni di inserimento e cancellazione comportino la traslazione verso l'alto o verso il basso dei record ricorrendo a delle pagine ausiliarie dette di **overflow:** ogni nuovo inserimento non avviene nella pagina ma in quella di overflow ad essa associata . Questo stratagemma, tuttavia, non è efficiente in termini di memoria e penalizza le operazioni di scansione che non sarebbero più sequenziali.

Ricapitolando: la struttura ordinata è indicata per l'accesso diretto e veloce ad un record e per le operazioni selezione. Non presenta particolari vantaggi sulle altre strutture per quanto concerne le operazioni di scansione dei file e rende poco efficienti e veloci le operazioni di inserimento/cancellazione.

Strutture ad accesso calcolato (hash)

Questa struttura organizzativa prevede che la posizione di un record all'interno di un file sia determinata dal valore restituito dall'applicazione di una funzione ad uno o più campi detti **chiave**. Come per la precedente struttura l'accesso ai record è **diretto** ed **associativo** ma non viene imposto alcun ordinamento.

La struttura del file è particolare: esso è suddiviso in B blocchi denominati ***bucket***. Ciascun bucket è, a sua volta, suddiviso in F ***slot*** della dimensione di un record. Ad ogni bucket viene associato un valore detto **hash**. Un bucket può essere composto da una o più pagine: la pagina principale o primaria, sempre presente, e le pagine di **overflow**, presenti solo ad avvenuto riempimento della pagina principale. Le pagine di overflow sono

collegate tra loro a formare le **catene di overflow**, la cui lunghezza non ha teoricamente limiti.

L'accesso associativo avviene in questo modo:

1. Si applica alla chiave un funzione f() che ne trasforma il valore in un intero positivo (*folding*);
2. Si applica al valore intero positivo restituito dalla predetta funzione f() una seconda funzione h() che lo mappa in valore intero positivo compreso nell'intervallo (0, B-1), con B numero di *bucket* in cui è suddiviso il file (*hashing*);
3. Si accede al record desiderato o per scansione all'interno del bucket individuato o per ulteriore applicazione della funzione *hash* (*hascing di ordine superiore*).

Da quanto detto appare chiaro che possono esistere più valori di una chiave che corrispondono allo stesso *bucket*, cioè che si mappano nello stesso valore della funzione *hash*. Questo fenomeno è detto *collisione* e non è da ritenersi negativo, se ben controllato, infatti, come abbiamo visto, ad ogni *bucket* è associata almeno una pagina di memoria ed, eventualmente, una catena di **overflow**.

Anche se le catene di overflow risolvono il problema delle collisioni, come contropartita causano un degrado prestazionale in quanto l'accesso diretto ad un record avviene, come già detto, o per scansione diretta della catena o tramite l'applicazione ricorsiva della tecnica *hash*.

E' importante, quindi, saper dimensionare bene questo tipo di organizzazione dei dati nei file. E' facile verificare che, all'aumentare del numero B di *bucket*, diminuisce la probabilità che si verifichino collisioni e, di conseguenza, che le catene di overflow rimangano di dimensioni limitate.

Infatti se **F** è il numero medio di record contenuti in una pagina, la probabilità $\mathbf{P_F}$ che avvengano più di F collisioni su un bucket e che quindi si generi una catena di overflow è data da:

$$P_F = 1 - \sum_{i=0}^{F} p(i)$$

dove p(i) rappresenta la probabilità che applicando la funzione hash sulle ***T*** tuple del file si verifichino ***i*** collisioni su un bucket tra B possibili:

$$p(i) = \binom{T}{i}\left[\left(\frac{1}{B}\right)^{i} * \left(1 - \frac{1}{B}\right)^{T-i}\right]$$

All'aumentare del parametro B, dunque, migliora l'efficienza in termini di velocità di accesso ai dati ma peggiora l'efficienza in termini di occupazione di memoria poiché si abbassa il cosiddetto **coefficiente di riempimento c del file**.

$$c = \frac{T}{B * F}$$

Questo coefficiente è un indice di distribuzione dei recod del file sui bucket. Da osservazioni risulta che il metodo hash raggiunge il miglior compromesso tra efficienza di velocità ed efficienza di impiego di memoria per un valore del coefficiente **c=0,8**. Dunque un buona scelta di B è:

$$B = \frac{T}{0,8 * F}$$

Il caricamento iniziale del file avviene attraverso la sua scansione completa. Le operazioni di cancellazione comportano la riorganizzazione dei record all'interno del bucket e, pertanto, sono poco efficienti. Le operazioni di inserimento non comportano riorganizzazioni dei record. Le operazioni di ricerca e di selezione del singolo record sono veloci ed efficienti se avvengono sui campi che costituiscono la chiave. Le selezioni su range di valori sono inefficienti rispetto alla scansione del file.

Le Strutture Ausiliarie

Alcuni metodi di accesso non prevedono una precisa organizzazione fisica dei record all'interno del file ma si basano su delle strutture dati ausiliarie per la localizzazione dei record. Le più importanti strutture ausiliarie sono gli **indici**, il cui funzionamento è simile a quello dell'indice di un libro: dato un valore di ricerca, detto chiave, che individui uno o più argomenti, si entra nell'indice e lo si esplora alla ricerca dei numeri delle pagine o dei numeri dei paragrafi del libro dove quegli argomenti sono trattati.

Anche in un file, come in un libro, le pagine ed i record sono numerati. Ogni pagina è contraddistinta da un valore univoco, detto ***Page ID***, ed ogni suo ***slot*** è contraddistinto da un valore unico all'interno della pagina, lo ***slot number***. Un record del file, occupando uno slot di una pagina, viene identificato univocamente dalla coppia di valori: ***<Page ID, Slot Number>***, denominata ***Record ID*** ovvero ***RID***. E', appunto, a questi identificatori che fanno riferimento le strutture ausiliarie per localizzare un record di un file.

L'importanza degli indici e di tutte le altre strutture dati ausiliarie è quello di poter consentire ad uno stesso file diversi metodi di accesso, indipendentemente dall'organizzazione interna dei record. Si è visto, infatti, che per ogni file non esiste un metodo di accesso migliore di un altro perché ciascuno di essi favorisce solamente alcuni tipi di operazioni. Ciascun metdodo di accesso, tuttavia, prevede una particolare forma di organizzazione dei dati nel file, scelta la quale sono preclusi gli altri metodi di accesso. Adottando gli indici è possibile, ad esempio, applicare ad un file organizzato secondo il modello Heap metodi di accesso simili al *Sort Merge* o all'*Hash Merge* perché questi non insistono direttamente sul file ma su opportune strutture dati ausiliarie ad esso associate.

Di indici ve ne sono di diversi tipi ma i più diffusi nei DBMS commerciali sono:

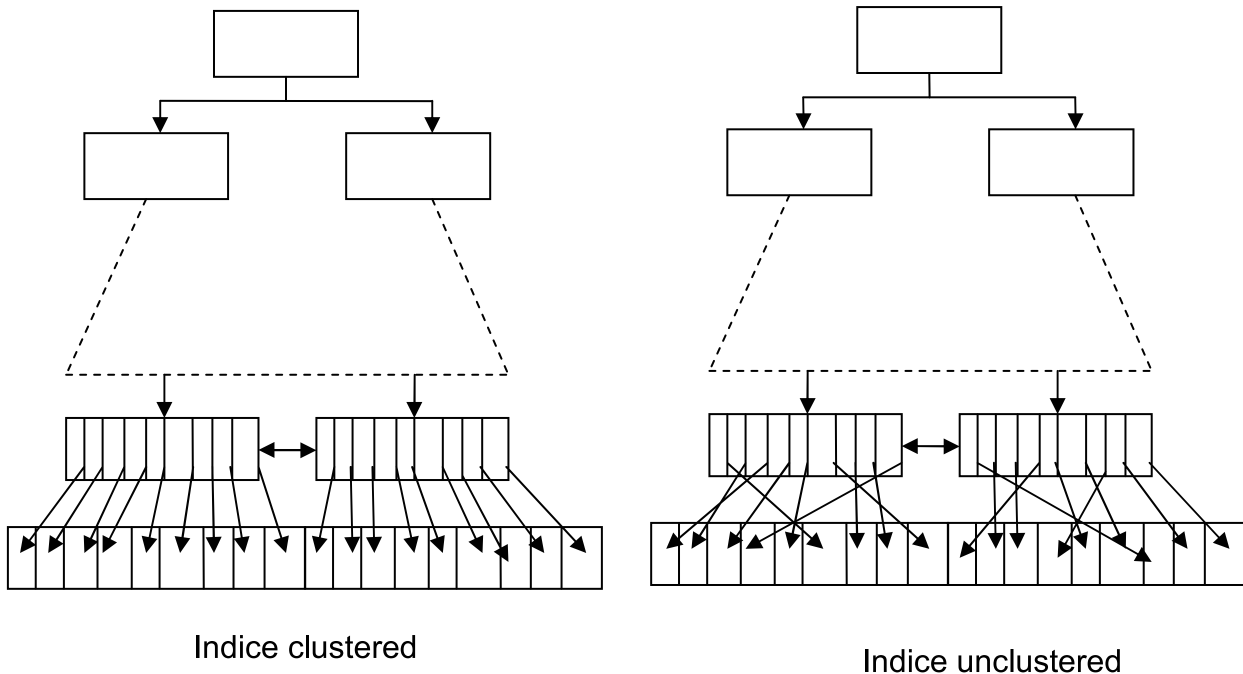
- Gli indici ISAM;
- Gli indici ad Albero, con particolare riferimento alle strutture B e B+;
- Gli indici Hash

<u>Proprietà degli indici</u>

Esistono diverse tipologie di indici che si diversificano tra loro per alcune proprietà:

1. **indici CLUSTERED ed UNCLUSTERED**: quando l'organizzazione e l'ordinamento delle registrazioni <K, Page ID> dell'indice (*data entries*) è identico a quello dei record del file allora si dice che l'indice è di tipo ***CLUSTERED***, altrimenti ***UNCLUSTERED***.

Nella seguente figura vengono messe a confronto le due tipologie di struttura



Indice clustered Indice unclustered

Ovviamente è possibile definire un solo indice Clustered per file mentre si possono definire molti indici unclustered. Gli svantaggi degli indici clustered sono evidenti: necessitano di operazioni di riorganizzazione ad ogni operazione di inserimento e/o di cancellazione perché tali operazioni sul file ordinato provocano un riordinamento dei record e quindi una variazione dei rispettivi RID. Il grosso vantaggio degli indici clustered è quello di agevolare le operazioni di selezione su range di valori e di accesso a più record con valori di chiave molto prossimi.

2. **Indici DENSI ed indici SPARSI:** un indice è detto **DENSO** se possiede almeno un elemento o registrazione (*data entry*) per ogni valore della chiave K presente nel file. Un indice è detto **SPARSO** se esiste un elemento o registrazione per ogni pagina del file. Si possono trovare indici CLUSTERED sia densi che sparsi, ma gli indici SPARSI possono solo essere CLUSTERED. Infatti non vi sarebbe alcun vantaggio nel definire un indice ordinato sui valori di un campo chiave K le cui registrazioni referenziano pagine del file che non rispettano quell'ordinamento. Per ogni file è possibile definire un solo indice **SPARSO** (clustered).

Gli indici ISAM

Gli indici ISAM (*Indexed Sequential Access Method*) sono strutture ausiliarie gerarchiche che consentono accessi associativi ai record in base al valore di uno o più campi o chiave.

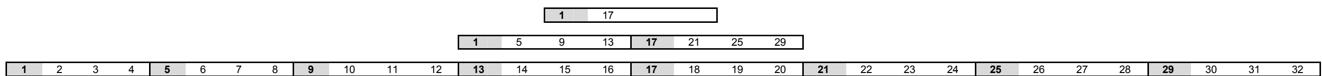
Supponiamo di avere un file di N record, ordinati sul valore di un campo chiave K, costituito da B pagine di memoria. Un metodo per velocizzare le operazioni di ricerca è quello di associare al file un file ausiliario, le cui pagine contengono registrazioni (*data entries*) del formato: < K, Page ID>, dove e Page ID è il riferimento ad una pagina del file e K è il primo valore del campo chiave, secondo l'ordinamento, registrato in quella pagina.

Chiamiamo F, o *Fan Out*, il numero di registrazioni < K, Page ID> contenute in una pagina di memoria.

Si intuisce che per accedere ad un record k_1 non è più necessario applicare la ricerca a tutto il file, bensì limitarla al file degli indici ed alla pagina contenente il record cercato. Il costo di una ricerca binaria su tutto il file è $Log_2(B)$ operazioni di lettura, mentre

il costo di una ricerca binaria attraverso un indice ISAM è pari a $Log_2(B/F) +1$ operazioni di lettura, infatti il numero di pagine del file degli indici è pari a B/F e quindi il costo per la ricerca binaria della pagina contenente il record cercato è $Log_2(B/F)$ al quale si deve, poi, aggiungere il costo di lettura della pagina tovata.

Se B/F, cioè il numero di pagine del file degli indici, è ancora molto grande, allora è possibile applicare al file degli indici un secondo file di indici di lunghezza pari a B/(F*F). Se anche questo numero fosse molto grande, è possibile ripetere n volte il procedimento fino ad ottenere un file di indici ragionevolmente corto. La dimensione ottimale per il file degli indici, è una pagina.



Come si vede si è ottenuta una struttura gerarchica a più livelli. La profondità della struttura, ovvero il numero di livelli, dipende dal *fan out* F e dal numero B di pagine del file: $d= Log_F(B)$.

Operazione di ricerca

Le operazioni di ricerca avvengono in modo molto efficiente, basta procedere, come già spiegato, dal file degli indici al più alto livello e scendere verso la pagina del file originario contenente il record cercato attraversando le pagine dei file di livello inferiore. Nell'esempio della precedente figura, per cercare il valore k=26, si eseguono in sequenza i seguenti pochi passi interattivi:

1. si accede alla pagina del file ad alto livello;
2. si accede alla pagina di livello 2 referenziata dall'indice K'=17;
3. Si accede alla pagina del file referenziata dall'indice K''=25.

Il costo in operazioni di I/O di una ricerca è pari al numero di pagine della struttura ISAM da leggere e, quindi, è pari al numero *d* di livelli da attraversare.

$$d= Log_F(B)$$

Tanto maggiore è il *Fan Out F*, tanto minore è la profondità *d* della struttura e quindi tanto più veloce è la ricerca.

Operazione di inserimento

L'operazione di inserimento di un record implica una preventiva operazione di ricerca per localizzare la pagina che gli corrisponde nell'ordinamento. Se questa presenta dello spazio disponibile, allora l'inserimento viene effettuato in quella pagina, altrimenti in una pagina di *overflow*. Nell'ISAM gli inserimenti non rispettano l'ordinamento imposto dalla chiave per mantenere alto il livello prestazionale in quanto, oltre alla dispendiosa operazione di riordino del file, vi è anche l'operazione di riorganizzazione dell'indice.

Operazione di cancellazione

L'operazione di cancellazione di un record implica una preventiva operazione di ricerca nell'indice ISAM e nelle eventuali pagine di overflow. Se la rimozione di un record lascia vuota una pagina di overflow, questa viene rimossa. La rimozione di un record, comunque, non implica la riorganizzazione del file e dell'indice

Osservazioni:

Per quanto visto negli ultimi tre paragrafi, le prestazioni degli indici ISAM degradano con il tempo se vengono interessati da frequenti operazioni di modifica del file. Gli ISAM non hanno meccanismi di riorganizzazione dinamica ed automatica della struttura dati ausiliaria e per questo vengono detti **statici**. Essi, pertanto, hanno bisogno di periodiche operazioni di manutenzione (rigenerazione) da parte di operatori esterni (amministratori) affinché conservino un buon livello di efficienza.

Gli indici ad Albero B tree e B+tree

Sono le strutture più usate nell'ambito dei DBMS commerciali. Esse consentono accessi associativi in base al valore di uno o più campi, detti **chiave (*key*)**, ma non vincolano la collocazione fisica delle tuple in posizioni specifiche del file.

Un albero consiste in un insieme di pagine, dette **nodi**, collegate tra loro attraverso dei puntatori in modo gerarchico. Alla base della gerarchia, cioè all'ultimo livello, vi sono delle pagine, denominate **foglie**, che non referenziano altri nodi ma contengono o referenziano i dati utili. Il nodo d'ingresso, cioè quello al livello più alto e dal quale partono tutte le operazioni di ricerca dei dati, è detto **radice**.

Ciascun nodo è una collezione ordinata di **F** record 'particolari' costituiti da un valore K_i della chiave di ordinamento e da un puntatore P_i ad un altro nodo dell'albero. Tra il valore della chiave ed il puntatore esiste una relazione: P_i punta al sottoalbero contenente i valori K_k della chiave maggiori di K_i e minori di K_{i+1}.

$$P_i : K_i \le K_K < K_{i+1}$$

Il puntatore P_o, in particolare, punta al sottoalbero i cui valori sono inferiori al valore K_1, mentre il puntatore P_F punta al sottoalbero i cui valori sono maggiori o uguali al valore K_F.

Dunque un nodo di un albero può puntare fino ad F sottoalberi. Il numero F dipende dalla dimensione della pagina, dalla dimensione dei puntatori e dalla dimensione dei valori della chiave e prende il nome di ***Fan-out***. Si può facilmente intuire che l'**efficienza di un indice ad albero è tanto maggiore quanto maggiore è F** perché minore è la profondità complessiva dell'albero.

Una delle caratteristiche delle strutture B tree e B+tree è quella di essere **bilanciate**, ovvero tutti i cammini dalla radice ad una foglia sono di ugual lunghezza, in modo da garantire tempi d'accesso alle informazioni costanti.

Esistono tre alternative per l'organizzazione dei dati nelle foglie di un albero:

1. Ogni elemento di una foglia è un record di dati utili (*key sequenced*) Questi indici sono sempre **CLUSTERED** e **DENSI**;
2. Ogni elemento di una foglia è un puntatore ad un record (<page ID, slot number>) di dati utili. Questi indici possono essere sia **DENSI** che **SPARSI**;
3. Ogni elemento di una foglia è un puntatore ad una pagina di dati utili, tutti aventi lo stesso valore della chiave K (*RID-list*). Questi indici possono solo essere **SPARSI**.

Operazione di ricerca

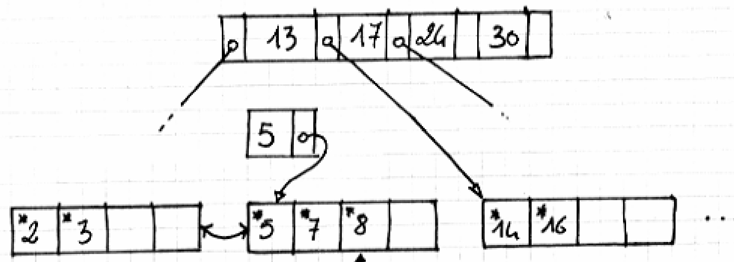
L'operazione di ricerca di un record all'interno di un albero, dato il relativo valore della chiave K, è semplice, basta confrontare, per ogni nodo attraversato a partire dalla radice, il valore della chiave di ricerca K con i valori della chiave contenuti nel nodo e procedere, scendendo di livello, verso il nodo puntato da P_o, se $K \le 1$, P_i se $K_i \le K < K_i+1$, P_F se $K > K_F$.

Operazione di Inserimento

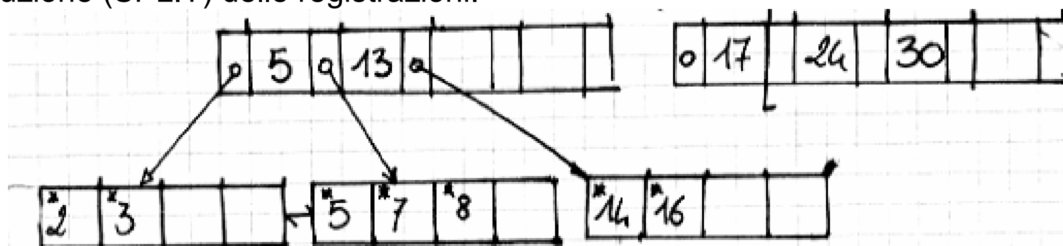
L'operazione d'inserimento prevede una preventiva operazione di ricerca per localizzare la posizione, all'interno del file, dove inserire il nuovo record. **Il requisito fondamentale per l'operazione d'inserimento è che l'albero rimanga bilanciato** e questo comporta l'esistenza di un meccanismo dinamico di bilanciamento. Inoltre è necessario che l'indice ottimizzi l'impiego della memoria mantenendo costante il coefficiente di riempimento delle pagine delle foglie e dei nodi. Se, ad esempio, una pagina potesse contenere 2d valori e 2d+1 puntatori e si volesse mantenere costante il coefficiente di riempimento a 0,5, allora nessun nodo dell'albero potrebbe contenere meno di d valori chiave e d+1 puntatori. Un nuovo nodo sarebbe, perciò, introdotto solo se fosse possibile ridistribuire gli elementi dei nodi adiacenti (*sibling*) in modo che, alla fine, ciascuno contenesse almeno d valori chiave e d+1 riferimenti. Questa operazione prende il nome di *SPLIT*.

L'inserimento non comporta particolari sforzi nel caso sia possibile inserire il nuovo valore K nella foglia dell'albero che gli corrisponde secondo l'ordinamento per la disponibilità di slot liberi. In caso contrario, o se non esiste già una foglia dove inserire il valore K, è necessario introdurne una nuova e bilanciare la struttura.

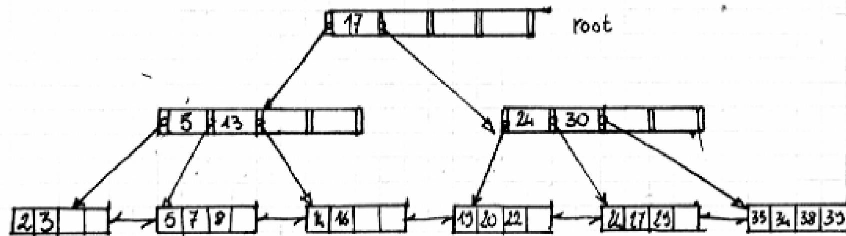
L'aggiunta di una nuova registrazione in una pagina satura (nell'esempio di figura K=8) può comportare l'introduzione di una nuova pagina foglia e la distribuzione (SPLIT) delle registrazioni della pagina satura su quella nuova.



L'operazione di suddivisione, che ha anche lo scopo di rendere costante il tasso medio di riempimento delle foglie, può comportare un nuovo assetto anche nei nodi di livello superiore, con funzione di ricerca, perché è possibile che si debba aggiungere una nuova registrazione (nell'esempio K=5). Anche in questo caso, se il nodo di livello superiore è saturo, si deve procedere all'introduzione di un nuovo nodo ed alla distribuzione (SPLIT) delle registrazioni.



Questa operazione di introduzione di nuovi nodi e di suddivisione (SPLIT) può propagarsi verso l'alto fino a raggiungere la radice e far aumentare la profondità dell'albero.



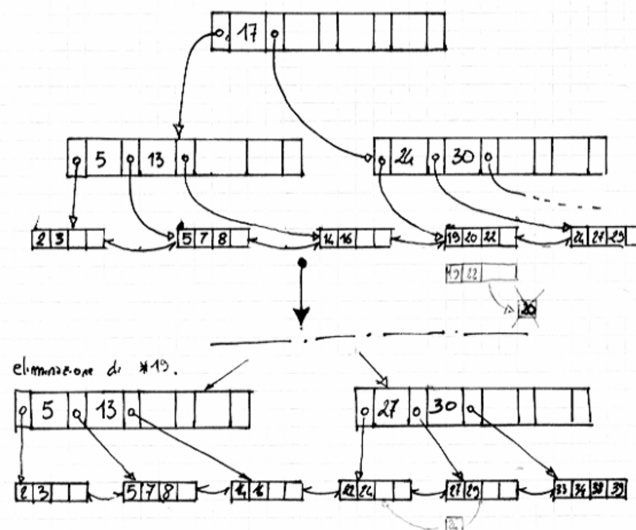
Operazione di cancellazione

Come per l'operazione di inserimento, anche per la cancellazione di un record dal file è necessario riorganizzare la struttura per mantenere sia il bilanciamento, sia l'efficienza di impiego delle pagine di memoria nella struttura. Se, pertanto, con la ripetuta cancellazione di record dalla base di dati, uno dei nodi venisse a contenere meno di d chiavi allora sarebbe possibile:

1) Ridistribuire i valori dei nodi adiacenti (*Sibling*) in modo che, alla fine, ciascuno di essi contenga almeno d valori di chiave e d+1 puntatori e sistemare, poi, i valori delle chiave ed i rispettivi puntatori nei nodi di livello superiore;

Esempio:

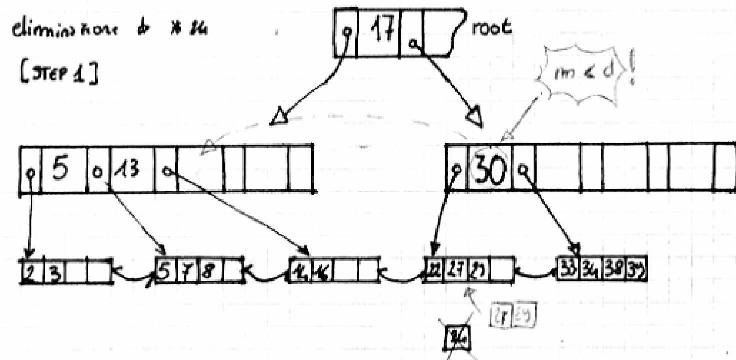
L'eliminazione dall'albero delle chiavi K'=20 e K''=19 non causa l'eliminazione di alcun nodo dell'albero ma semplicemente lo spostamento del valore K'''=24 da un nodo adiacente (*Sibling*) e la sostituzione del valore K=24 col valore K=27 nel nodo di livello superiore (*Copy Up*).



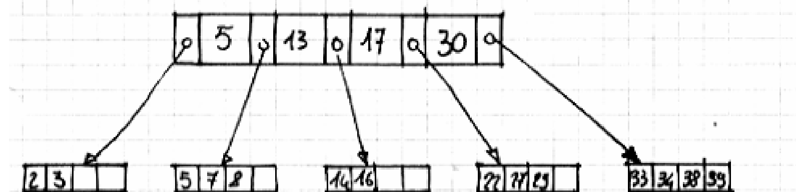
2) Nel caso in cui, a seguito di ripetute cancellazioni, non sia ulteriormente possibile distribuire il contenuto di più nodi adiacenti (*Sibling*) in modo da mantenere superiore a 0,5 il loro coefficiente di riempimento, si deve procedere all'eliminazione di un nodo, collocando nei nodi adiacenti i rimanenti valori della chiave e riorganizzando i valori dei nodi di livello superiore. Questa operazione prende il nome di **MERGE** ed è la duale dell'operazione di split:. L'operazione di merging può, ovviamente, interessare anche i nodi di livello superiore e risalire l'albero fino alla radice causandone una riduzione della profondità.

Esempio:

A partire dall'albero della precedente figura, si elimini il valore K=24, si rimuova la foglia che lo conteneva e si sposti il valore K=22 nella foglia adiacente. Le foglie rimanenti risultano, così, avere un coefficiente di riempimento maggiore di 0,5 e l'albero, nel complesso, rimane bilanciato. E' necessario, tuttavia, correggere le informazioni contenute nel nodo di livello superiore: il valore K=27 non ha più ragione d'essere presente e quindi è possibile procedere alla sua rimozione.



Quest'ultima operazione, tuttavia, lascia il nodo di livello superiore con un coefficiente di riempimento minore di 0,5. Non essendo praticabile la redistribuzione dei valori di chiave dai nodi adiacenti (*sibling*) è necessario sopprimere il nodo ed effettuare nuovamente l'operazione di **merging,** aggiustando i valori della chiave ed i puntatori del nodo di livello superiore. Nel caso in esempio il valore K=17 non ha più ragione di permanere nel nodo di livello superiore e deve, quindi, essere spostato (*Move Down*) nel nodo di livello inferiore. Questa operazione potrebbe anche portare ad una riduzione della profondità dell'albero



In considerazione di quanto detto sopra, gli indici ad albero consentono operazioni di ricerca ed accesso molto veloci al singolo dato (selezioni su predicato di eguaglianza). Gli inserimenti e le cancellazioni possono dare luogo a riorganizzazioni della struttura e quindi penalizzare le prestazioni. Le selezioni su range di valori sono molto onerose.

Per rendere efficienti anche le operazioni di selezione su range di valori, le strutture ad albero B+ hanno le foglie doppiamente collegate tra loro in modo che, a partire da un valore, sia possibile effettuare velocemente la scansione del file secondo l'ordine imposto dall'indice. Questa è la differenza tra gli indici B tree e gli indici B+tree.

Un ulteriore accorgimento per ottimizzare gli accessi alle foglie negli alberi B tree è quello di associare nei nodi intermedi due puntatori per ogni valore K della chiave:uno che punta al rispettivo sottoalbero ed uno che punta direttamente alla foglia corrispondente al valore K della chiave. In questo modo è possibile concludere la ricerca senza percorrere tutti i livelli e si risparmia spazio nelle pagine dell'indice perché ogni valore della chiave è presente nell'albero al più una sola volta.

Indicizzazione Hash