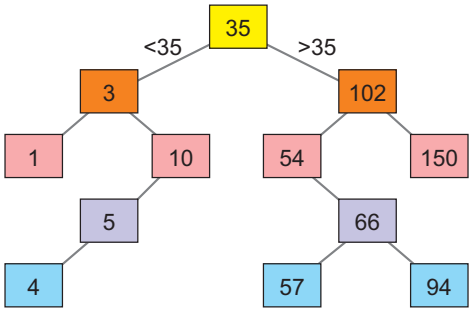
3. Alberi binari e tecniche di hashing per la gestione delle chiavi

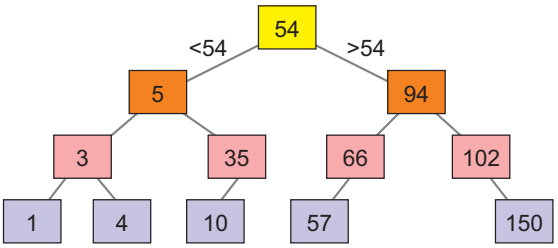
Alberi binari

Gli indici di un file possono essere trattati in modo efficiente con tecniche che si basano sull'uso di **alberi binari** per le chiavi. La ricerca infatti diventa più veloce se le chiavi vengono mantenute in ordine, in modo da poter utilizzare il metodo della ricerca binaria.

ORDINE DI INSERIMENTO DELLE CHIAVI
35 102 3 54 66 10 94 1 5 4 57 150



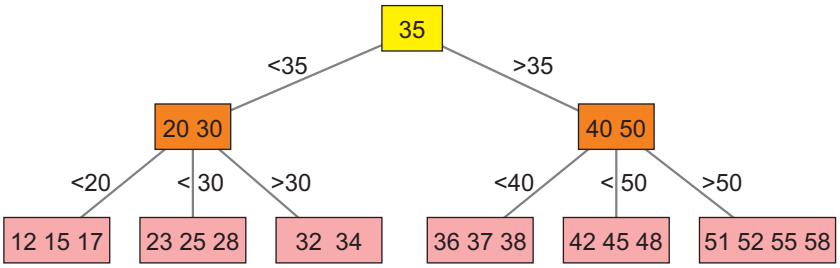
Per ottimizzare la ricerca è opportuno che l'albero si mantenga **bilanciato**: il bilanciamento consiste nel trasformare l'albero in modo da ottenerne uno equivalente, contenente cioè gli stessi dati, che abbia profondità minima, cioè con un numero basso di livelli gerarchici.



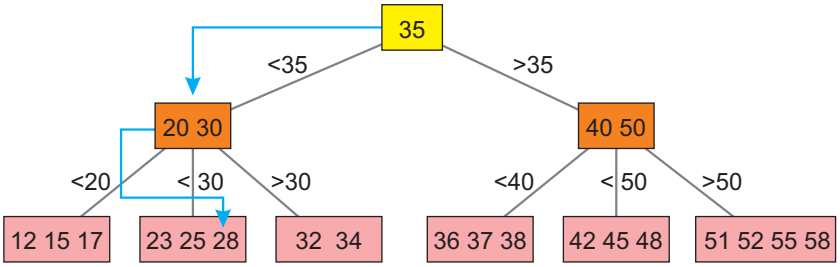
Per diminuire il numero di accessi nella struttura ad albero, nella ricerca delle chiavi, si usano alberi generali suddividendoli in sottoalberi, detti *pagine*, in modo tale che ogni accesso a una pagina richieda un solo accesso al disco. Con l'inserimento di nuove chiavi la struttura ad albero si espande e la crescita deve essere mantenuta sotto controllo per garantirne il bilanciamento. A queste esigenze risponde la struttura detta **B-tree** (o *Balanced tree,* albero bilanciato), che può essere considerata una generalizzazione degli alberi binari bilanciati.
Un B-tree di ordine *n* possiede le seguenti caratteristiche:

1. ogni pagina, contiene al massimo *2n* chiavi con i relativi puntatori ai record;
2. ogni pagina, esclusa la radice, deve contenere almeno *n* chiavi con i relativi puntatori ai record;
3. ogni pagina può essere una pagina terminale (foglia), oppure avere *m+1* discendenti, essendo *m* il numero delle sue chiavi;
4. tutte le pagine foglia hanno lo stesso livello gerarchico nella struttura ad albero.

Per esempio un albero B-tree di ordine 2 viene illustrato nella figura seguente:



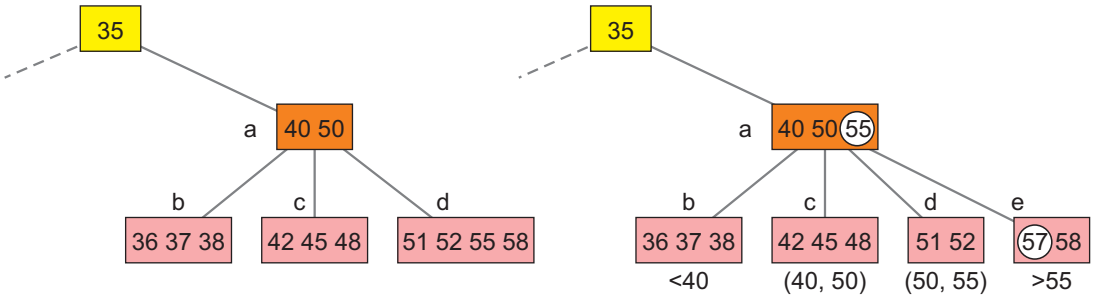
Si consideri per esempio la pagina contenente le due chiavi che hanno i valori 20 e 30: essa ha tre pagine come discendenti che contengono rispettivamente le chiavi minori di 20, comprese tra 20 e 30, e maggiori di 30.
La ricerca di una chiave nella struttura B-tree avviene come per gli alberi binari: a partire dalla radice vengono attraversati i nodi nei sottoalberi finché la chiave viene trovata oppure finché la ricerca termina senza successo. Se la chiave cercata è compresa tra quelle della pagina considerata, la ricerca si conclude in modo positivo (nella pagina oltre al valore della chiave è memorizzato un puntatore al record), altrimenti si va a esaminare tra le pagine discendenti quella corrispondente all'intervallo entro il quale è compresa la chiave richiesta.
Con riferimento all'esempio precedente, se la chiave da cercare ha valore 28, il percorso è il seguente:



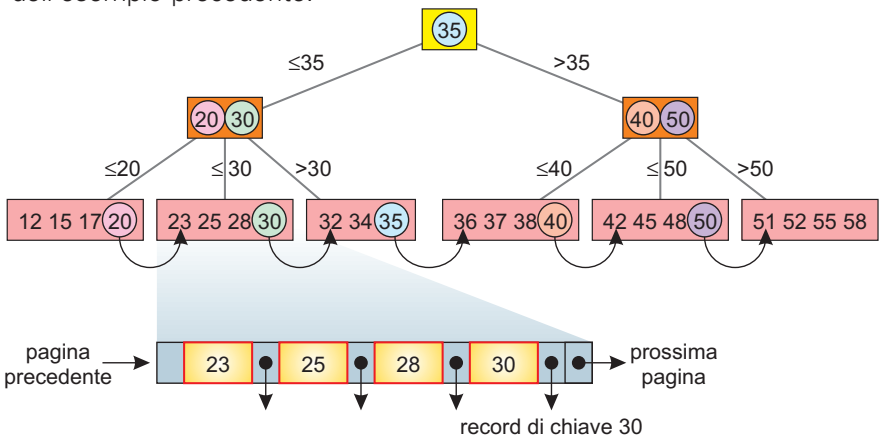
Nella struttura B-tree l'inserimento di una nuova chiave è un'operazione che deve avere l'obiettivo di mantenere l'albero bilanciato. Può accadere che l'inserimento richieda l'allocazione di una nuova pagina, nel caso in cui la pagina adatta a contenere la nuova chiave sia piena perché contiene già *2n* elementi.

Per esempio, se nell'albero-B precedente si deve inserire la nuova chiave 57, si procede nel seguente modo:

- la chiave 57 non è presente, ma non può essere messa nella pagina contrassegnata dalla lettera **d** perché è piena;
- la pagina **d** viene divisa in due pagine, creando una nuova pagina **e**;
- le chiavi vengono suddivise in parti uguali tra le due pagine e l'elemento centrale (55) viene spostato alla pagina di livello superiore.

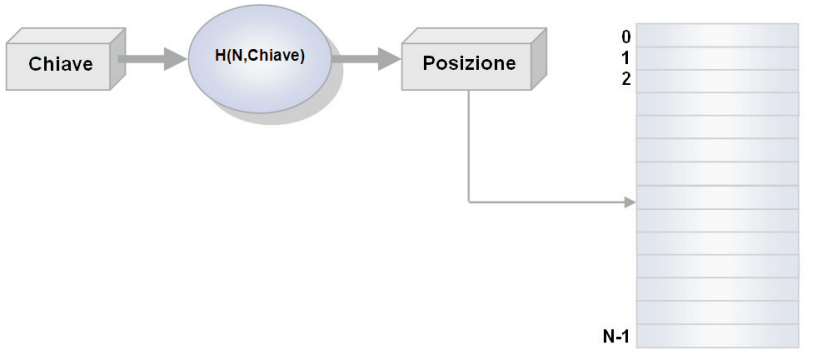


In questo modo vengono conservate le caratteristiche della struttura B-tree. Successivi inserimenti possono causare il riempimento anche della pagina di livello superiore, secondo un meccanismo di suddivisione delle pagine che può arrivare anche alla radice dell'albero: la struttura si espande così in modo dinamico, e allo stesso tempo controllato, al crescere del numero dei record memorizzati nel file.
La struttura B-tree dell'esempio precedente permette l'individuazione della posizione di un record accedendo a non più di tre pagine dell'albero: per esempio il record con chiave di valore 35 viene individuato al primo tentativo, quello con chiave di valore 40 al secondo, mentre il record con chiave 45 richiede l'accesso a tre pagine. È piuttosto complicato però accedere sequenzialmente ai record ordinati secondo il valore della chiave per rispondere, per esempio, alla seguente richiesta: "elencare tutti i record con chiave maggiore di 30".
L'organizzazione **B+-tree** permette di ovviare a questo inconveniente al prezzo di una certa ridondanza delle informazioni. In un albero B^+ solo i nodi foglia contengono puntatori ai record: pertanto tutti i valori delle chiavi nei nodi intermedi devono essere riportati negli opportuni nodi foglia. Inoltre le foglie dell'albero sono tutte collegate tra di loro per permettere la lettura sequenziale dei record. La figura mostra la struttura dei nodi foglia e l'albero B^+ corrispondente al B-tree dell'esempio precedente.



Le tecniche di hashing

I file con organizzazione random, come abbiamo visto, contengono record che vengono identificati attraverso la posizione che essi occupano all'interno del file.
Volendo associare a ciascun record un valore chiave, di norma alfanumerico, occorre determinare una funzione che possa trasformare la chiave in un numero. Questa tecnica di indicizzazione si chiama *metodo di trasformazione della chiave* o metodo **hash:** una funzione *hash*, avente come argomento una chiave, deve restituire la posizione del record corrispondente, con l'obiettivo di distribuire nel modo più uniforme possibile i record all'interno del file.



File ad accesso diretto

La funzione *hash* calcola un numero di record sulla base del valore della chiave e del massimo numero di record previsti per il file, cioè trasforma l'intervallo dei valori della chiave (che può essere anche molto vasto) in un numero compreso tra 0 e NR-1, dove NR è il numero di record nel file.
Poiché il calcolo dipende dal numero massimo di record per il file, in fase di creazione del file, occorre specificare qual è il numero massimo dei record.
Una semplice funzione *hash* utilizza un numero intero ricavato dalla chiave, divide questo intero per il numero massimo di record e usa il resto della divisione come valore restituito dalla funzione.
Per esempio i valori delle chiavi possono essere codici numerici compresi tra 1000 e 3000, anche se non tutti i numeri vengono utilizzati. Supponendo che i record debbano essere memorizzati in un file di 100 record, la funzione *hash* divide il codice numerico per 100, ottenendo come resto un numero compreso tra 0 e 99 che identifica la posizione nel file:

posizione = codice mod 100.

Nel caso di chiavi alfanumeriche si costruiscono funzioni *hash* basate sul valore dei codici ASCII (o di un'altra codifica) dei caratteri che compongono la chiave. Per esempio, supponiamo che la chiave sia composta da 4 caratteri alfabetici; si considera allora la somma dei codici ASCII dei caratteri componenti la chiave e si divide per il numero massimo di record presenti nel file ottenendo un numero che fornisce la posizione del record nel file, in fase di scrittura e di ricerca. Applichiamo questa tecnica ai record del file anagrafe usato in precedenza e consideriamo la prima delle 15 chiavi dei record che compongono il file: "Anza". Supponiamo di voler distribuire i record in un *hash* file con 23 record:

Eseguiamo il calcolo per la chiave "Lore"

Quando un record viene aggiunto in un file gestito con una tecnica hashing, il software di gestione dei file tenta di mettere il record nella posizione calcolata dalla funzione *hash*. Se quella posizione è già occupata da un record, accade una **collisione** e il record deve essere messo da qualche altra parte.
Calcoliamo, per esempio, la posizione di inserimento del record di chiave "Cava":

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

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

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

$$15 \times 1{,}3 = 19{,}5 \text{ arrotondato a } 23$$

si ottiene 23 come dimensione suggerita per il file. Il suggerimento, che ha lo scopo di ridurre il numero di possibili collisioni, è frutto di considerazioni di tipo statistico sulle prestazioni delle funzioni di *hashing* e, conseguentemente, non esclude che per particolari distribuzioni di valori delle chiavi si manifestino molte collisioni. Per esempio: modificando di poco l'algoritmo che abbiamo utilizzato nell'esercizio e applicandolo ai 15 record del file si evidenziano addirittura 8 collisioni.

Ci sono diverse tecniche per la **gestione delle collisioni.**

Con il metodo della **scansione lineare**, il record viene collocato nella prima posizione libera successiva a quella calcolata. Se viene raggiunta la fine del file, la ricerca continua ripartendo dall'inizio del file sino a trovare un posto libero.

Il procedimento ha fine nell'ipotesi di inserire un numero di record inferiore alla dimensione massima del file creato.

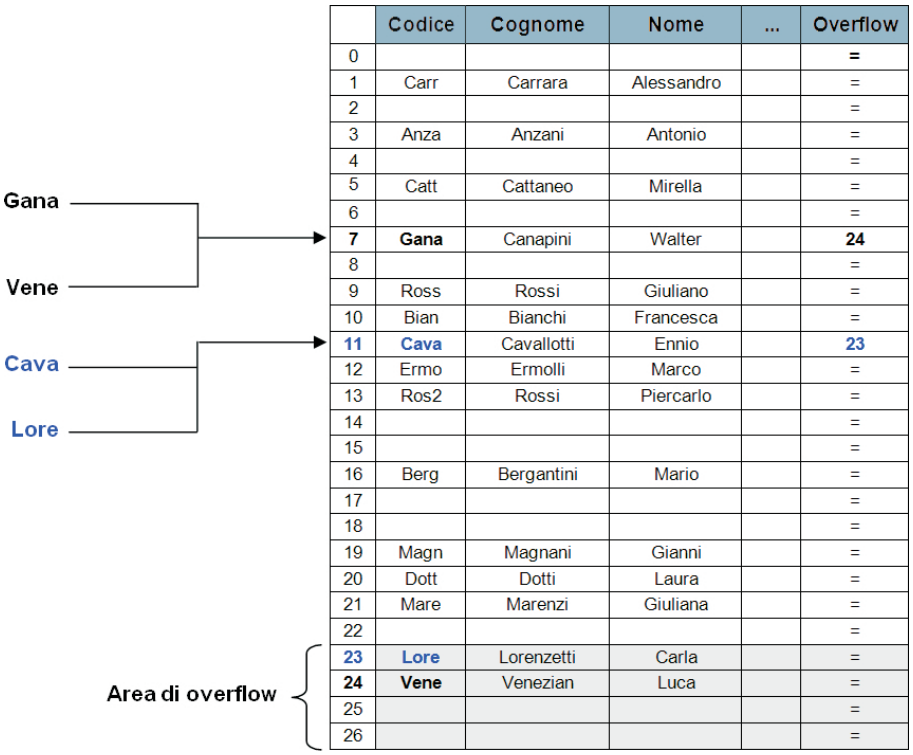
Per esempio supponiamo di inserire i record del file in ordine di chiave. Iniziamo dal record di chiave "Anza" inserito in posizione 3, seguito da "Berg" in posizione 16, "Bian" nel posto 10 e così via.

Va tutto bene sino a "Lore" che trova il posto 11 occupato. Con il metodo della scansione lineare si cerca di collocare "Lore" in posizione 12 (già occupata) e poi nel posto 13 che, essendo libero, viene occupato. Risolto un problema se ne presenta poco dopo un altro, non previsto, quando si cerca di allocare il record di chiave "Ros2" a cui competerebbe proprio il posto 13. Avevamo previsto due collisioni e invece ne troviamo tre.

Quindi con il metodo della scansione lineare è possibile che si creino lunghe catene di record adiacenti. Questo può rallentare l'accesso ai record in modo significativo, dal momento che sarebbe necessario leggere ciascun record nella catena per trovare quello desiderato.

Un metodo alternativo è rappresentato dalla **scansione quadratica**: viene calcolato il numero di record, se il posto è già occupato si va a controllate il primo posto successivo, poi il quarto, il nono e così via sino a un opportuno criterio di arresto. Questo metodo tende a distribuire le collisioni prodotte dalla funzione *hash* attraverso tutto il file, piuttosto che creare accumuli attorno a certe posizioni.

Una terza soluzione, molto diffusa nelle implementazioni di indicizzazione con funzione di *hashing*, consiste nel creare un'area separata, detta anche **area di overflow**, adatta a contenere i record che hanno valori di chiave corrispondenti a situazioni di collisione con le chiavi già esistenti.
Applicandola all'esempio precedente si giungerebbe alla situazione illustrata nella figura.



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