Strutture di accesso ai dati: B^+-tree

ALBERTO BELUSSI

SECONDA PARTE
ANNO ACCADEMICO 2011-2012

Osservazione

- Quando l'indice aumenta di dimensioni, non può risiedere sempre in memoria centrale: di conseguenza deve essere gestito in memoria secondaria.
- È possibile utilizzare un FILE SEQUENZIALE ORDINATO per rappresentare l'indice in mem. sec.
- Le prestazioni di accesso a tale struttura fisica a fronte di inserimenti/cancellazioni tendono a degradare e richiedono frequenti riorganizzazioni. Inoltre è disponibile solo l'accesso sequenziale.
- Per superare il problema si introducono per gli indici strutture fisiche diverse.
- Tra queste analizzeremo: le **strutture ad albero** e le **strutture ad accesso calcolato**.

B^+-tree: caratteristiche

Caratteristiche generali dell'indice B^+-tree:

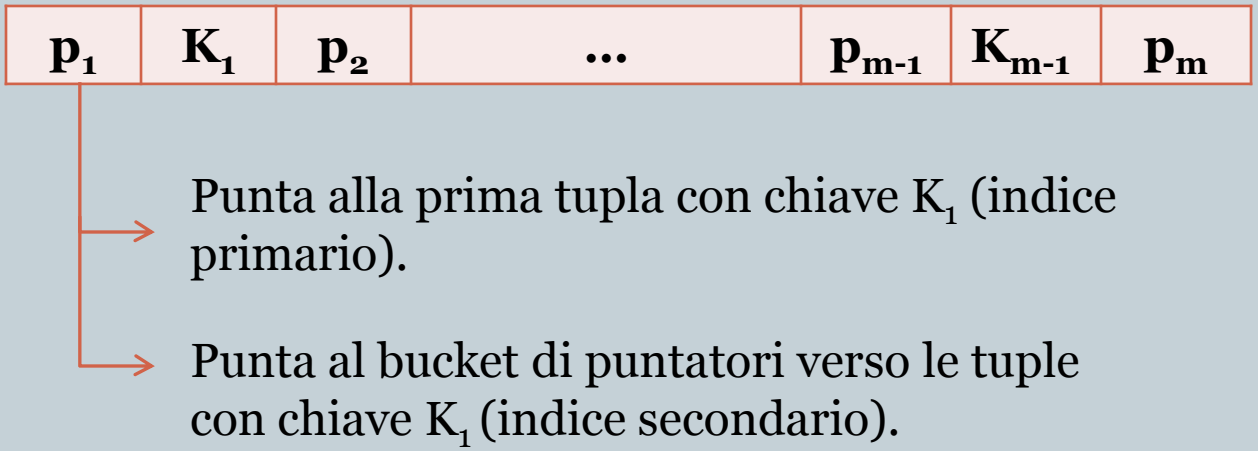
- è una struttura ad albero;
- ogni nodo corrisponde ad una pagina della memoria secondaria;
- i legami tra nodi diventano puntatori a pagina;
- ogni nodo ha un numero elevato di figli, quindi l'albero ha tipicamente pochi livelli e molti nodi foglia;
- l'albero è bilanciato: la lunghezza dei percorsi che collegano la radice ai nodi foglia è costante;
- inserimenti e cancellazioni non alterano le prestazioni dell'accesso ai dati: l'albero si mantiene bilanciato.

B$^+$-tree: struttura

Struttura di un B$^+$-tree (fan-out = n)

NODO FOGLIA

- può contenere fino a (n-1) valori della chiave di ricerca e fino a n puntatori.



$m \leq n$

$i<j \Rightarrow K_i<K_j$

- variante: al posto dei valori chiave il nodo foglia contiene direttamente le tuple (struttura integrata dati/indice)

B^+-tree: struttura

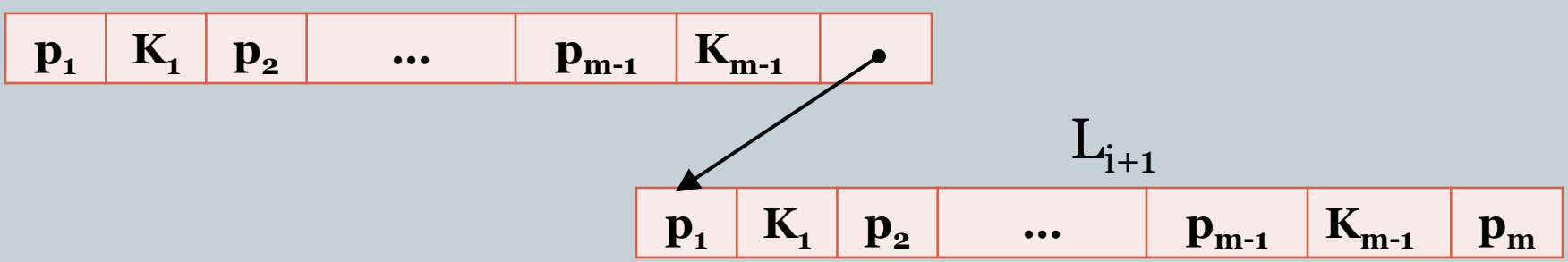
Struttura di un B^+-tree (fan-out = n)

NODO FOGLIA (vincolo di ordinamento)

- I nodi foglia sono ordinati. Inoltre, dati due nodi foglia L_i e L_j con i<j risulta:

$$\forall K_t \in L_i: \forall K_s \in L_j: K_t < K_s$$

- Il puntatore p_m del nodo L_i punta al nodo L_{i+1} se esiste.

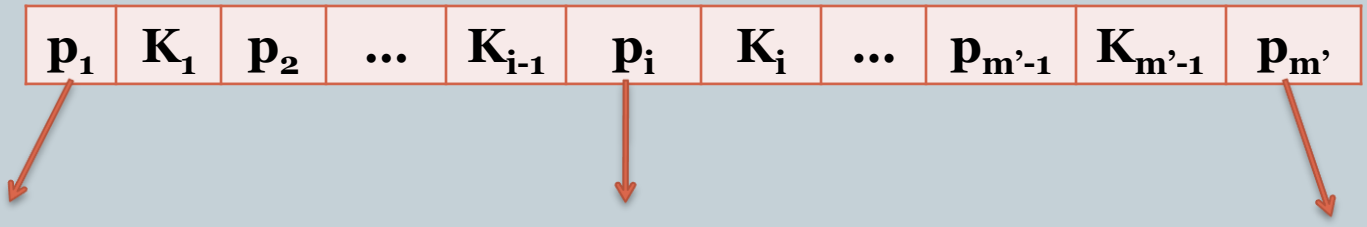


B$^+$-tree: struttura

Struttura di un B$^+$-tree (fan-out = n)

NODO INTERMEDIO

- può contenere fino a n puntatori a nodo.



$m' \leq n$

Punta al sottoalbero con chiavi k tali che:
$k < K_1$

Punta al sottoalbero con chiavi k tali che:
$K_{i-1} \leq k < K_i$

Punta al sottoalbero con chiavi k tali che:
$K_{m'-1} \leq k$

B⁺-tree: vincoli di riempimento (regole di bilanciamento)

Struttura di un B⁺-tree (fan-out = n)

NODO FOGLIA (vincolo di riempimento)

- Ogni nodo foglia contiene un numero di valori chiave *#chiavi* vincolato come segue:

$$\lceil (n-1)/2 \rceil \leq \#chiavi \leq (n-1)$$

NODO INTERMEDIO (vincolo di riempimento)

- Ogni nodo intermedio contiene un numero di puntatori *#puntatori* vincolato come segue (non vale per la radice):

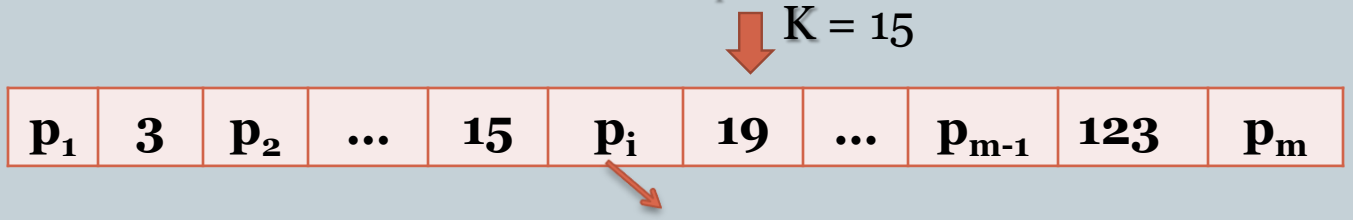
$$\lceil n/2 \rceil \leq \#\ puntatori \leq n$$

Arrotondamento all'intero superiore più vicino

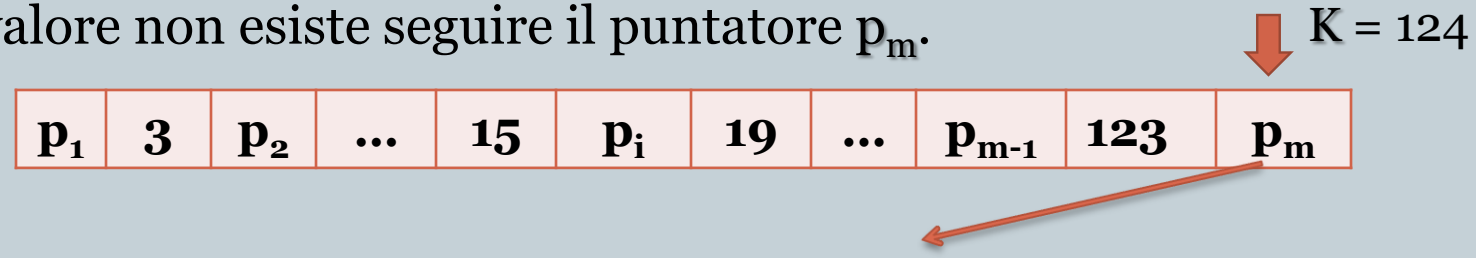
B$^+$-tree: operazioni

Ricerca dei record con chiave K

- Passo 1 – Cercare nel nodo (radice) il più piccolo valore di chiave maggiore di K.
  Se tale valore esiste (supponiamo sia K_i) allora seguire il puntatore p_i.



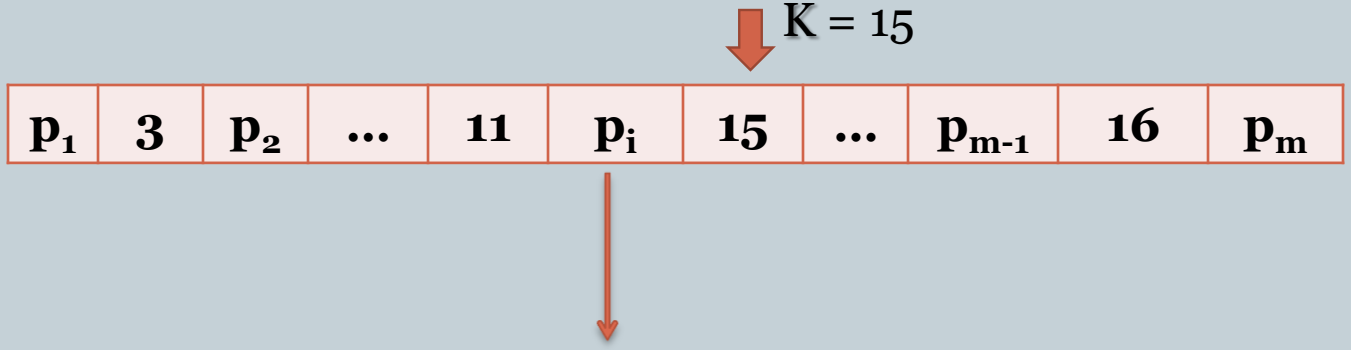
Se tale valore non esiste seguire il puntatore p_m.



B$^+$-tree: operazioni

Ricerca dei record con chiave K

- Passo 2 – Se il nodo raggiunto è un nodo foglia cercare il valore K nel nodo e seguire il corrispondente puntatore verso le tuple, altrimenti riprendere il passo 1.



B$^+$-tree: profondità dell'albero

Osservazione

- Il costo di una ricerca nell'indice, in termini di numero di accessi alla memoria secondaria, risulta pari al numero di nodi acceduti nella ricerca.
- Tale numero in una struttura ad albero è pari alla profondità dell'albero, che nel B$^+$-tree è indipendente dal percorso ed è funzione del fan-out ***n*** e del numero di valori chiave presenti nell'albero ***#valoriChiave***:

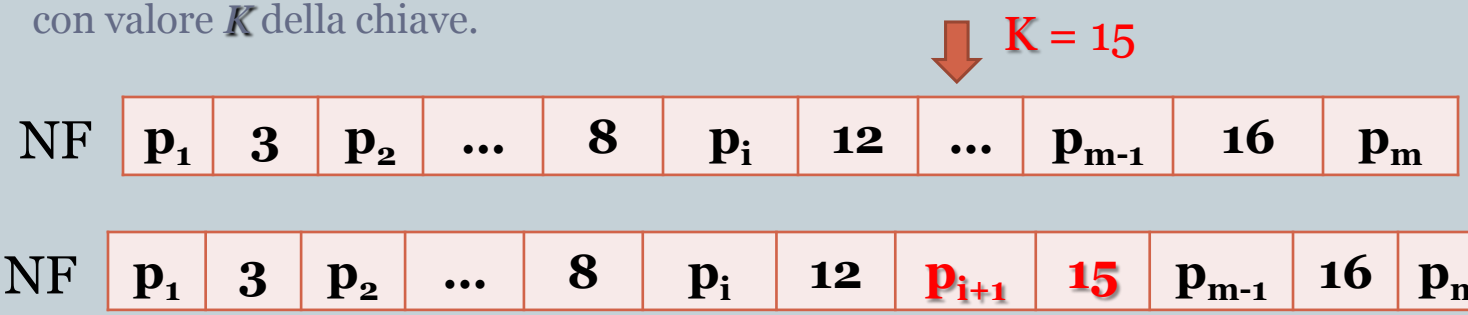
$$prof_{B+tree} \leq 1 + \log_{\lceil n/2 \rceil} \left(\frac{\#valoriChiave}{\lceil (n-1)/2 \rceil} \right)$$

B$^+$-tree: operazioni

Inserimento di un nuovo valore della chiave K

- Passo 1: ricerca del nodo foglio NF dove il valore *K* va inserito
- Passo 2: se *K* è presente in NF, allora:
  - Indice primario: nessuna azione
  - Indice secondario: aggiornare il bucket di puntatori

  altrimenti, inserire *K* in NF rispettando l'ordine e:
  - Indice primario: inserire puntatore alla tupla con valore *K* della chiave
  - Indice secondario: inserire un nuovo bucket di puntatori contenente il puntatore alla tupla con valore *K* della chiave.



se non è possibile inserire *K* in NF, allora eseguire uno **SPLIT** di NF.

B^+-tree: operazioni

Inserimento di un nuovo valore della chiave K

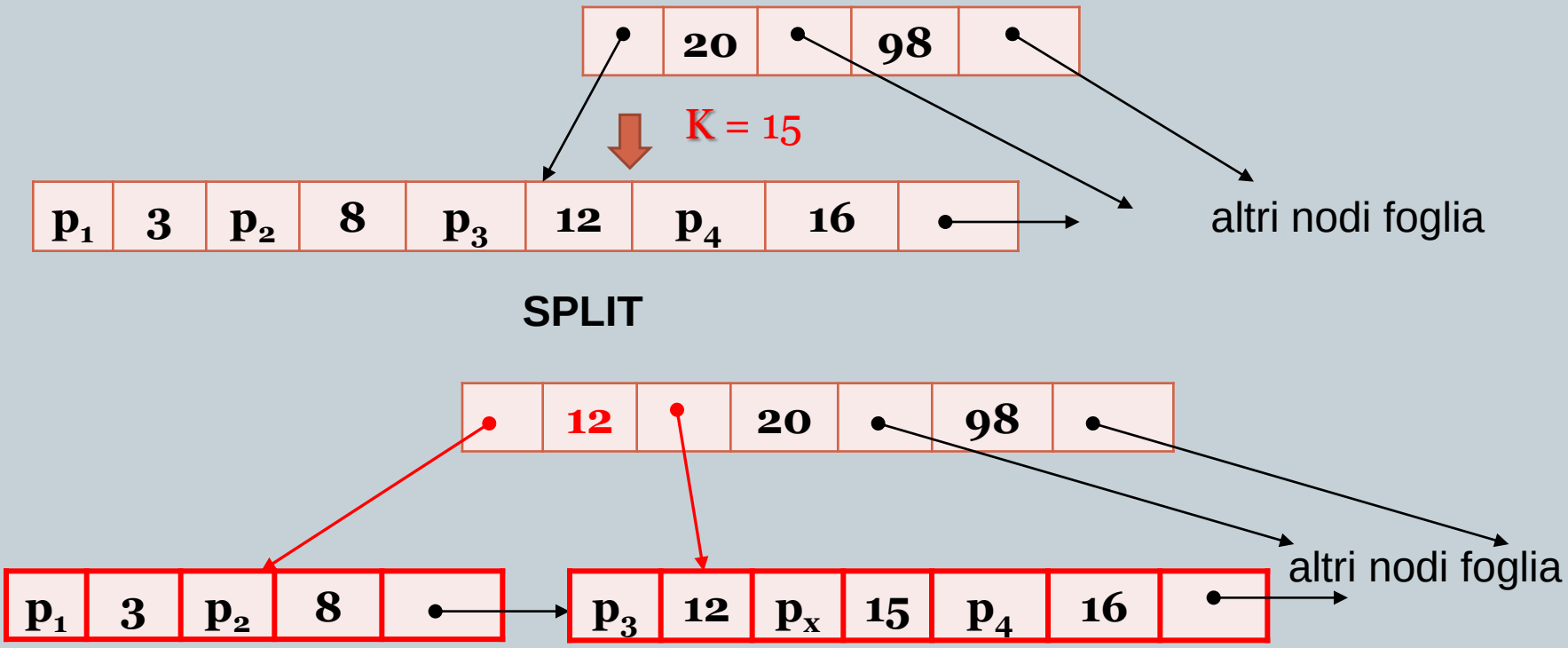
SPLIT di un nodo foglia

- Nel nodo da dividere esistono **n** valori chiave, si procede come segue:
  - Creare due nodi foglia;
  - Inserire i primi $\lceil (n-1)/2 \rceil$ valori nel primo;
  - Inserire i rimanenti nel secondo;
  - Inserire nel nodo padre un nuovo puntatore per il secondo nodo foglia generato e riaggiustare i valori chiave presenti nel nodo padre.
  - Se anche il nodo padre è pieno (n puntatori già presenti) lo SPLIT si propaga al padre e così via, se necessario, fino alla radice.

B$^+$-tree: operazioni

Inserimento di un nuovo valore della chiave K

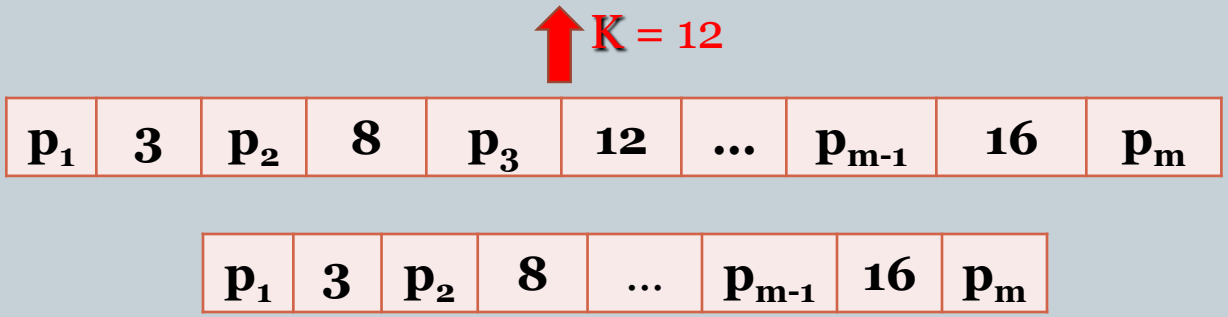
Esempio di SPLIT di un nodo foglia (fan-out = 5)



B$^+$-tree: operazioni

Cancellazione di un valore della chiave K

- Passo 1: ricerca del nodo foglia NF dove il valore *K* va cancellato
- Passo 2: cancellare *K* da NF insieme al suo puntatore:
  - Indice primario: nessuna azione
  - Indice secondario: liberare il bucket di puntatori



se dopo la cancellazione di *K* da NF viene violato il vincolo di riempimento minimo di NF, allora eseguire un MERGE di NF.

B^+-tree: operazioni

Cancellazione di un valore della chiave K

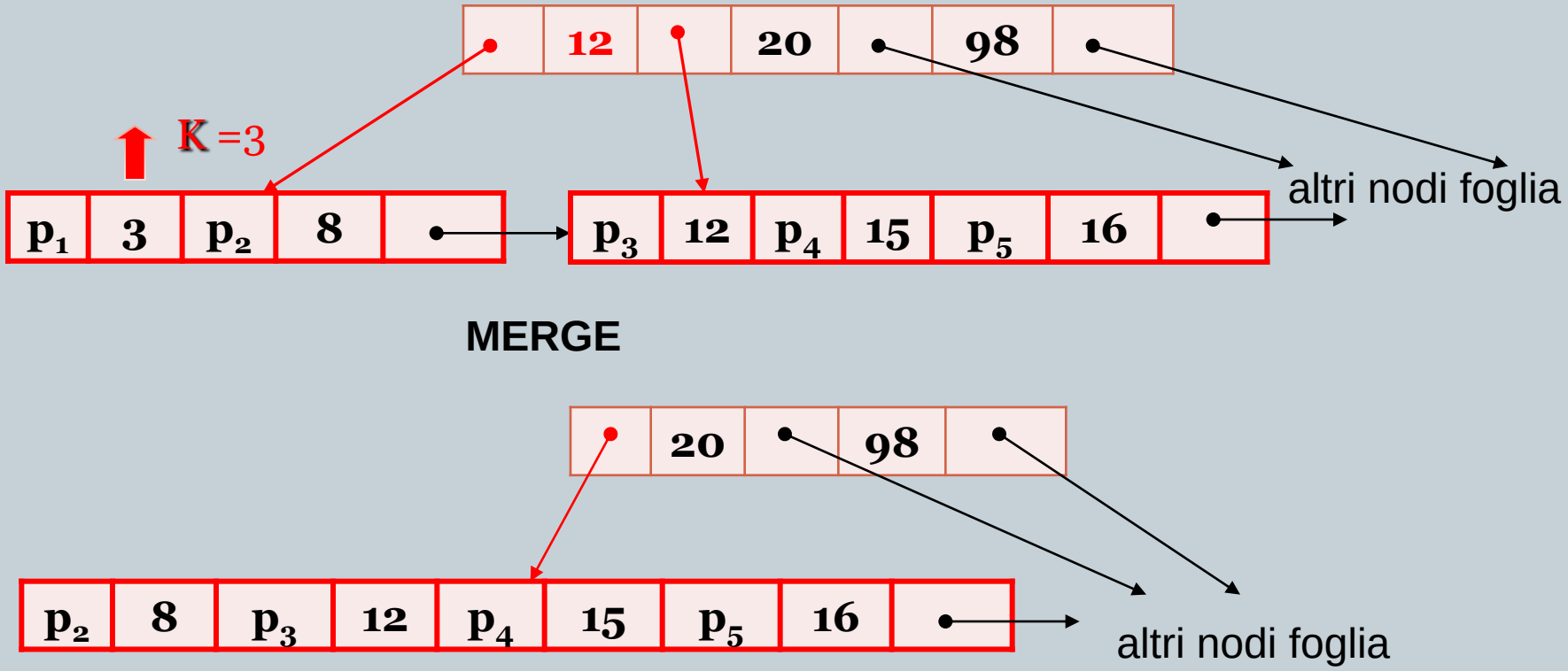
MERGE di un nodo foglia

- Nel nodo da unire esistono $\lceil (n-1)/2 \rceil - 1$ valori chiave, si procede come segue:
  - Individuare il nodo fratello adiacente da unire al nodo corrente;
  - Se i due nodi hanno complessivamente al massimo n-1 valori chiave, allora
    - si genera un unico nodo contenente tutti i valori
    - si toglie un puntatore dal nodo padre
    - si aggiustano i valori chiave del nodo padre
  - Altrimenti si distribuiscono i valori chiave tra i due nodi e si aggiustano i valori chiave del nodo padre
  - Se anche il nodo padre viola il vincolo minimo di riempimento (meno di n/2 puntatori presenti), il MERGE si propaga al padre e così via, se necessario, fino alla radice.

B$^+$-tree: operazioni

Cancellazione di un valore della chiave K

Esempio di MERGE di un nodo foglia (fan-out = 5)



Strutture ad accesso calcolato (hashing)

Caratteristiche generali delle strutture ad accesso calcolato:

- si basano su una funzione di hash che mappa le chiavi sugli indirizzi di memorizzazione delle tuple nelle pagine dati della memoria secondaria;

$$h: K \rightarrow B$$

K: dominio delle chiavi, B: dominio degli indirizzi

- Uso pratico di una funzione di hash
  - Si stima il numero di valori chiave che saranno contenuti nella tabella;
  - Si alloca un numero di bucket di puntatori (B) uguale al numero stimato;
  - Si definisce una funzione di FOLDING che trasforma i valori chiave in numeri interi positivi:

$$f: K \rightarrow Z^+$$

  - Si definisce una funzione di HASHING:

$$h: Z^+ \rightarrow B$$

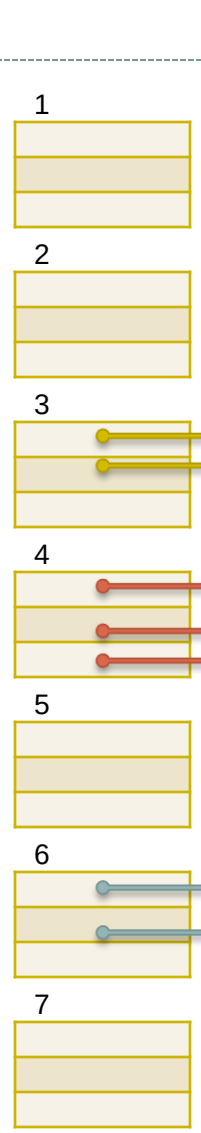
Strutture ad accesso calcolato (hashing)

Caratteristica di una buona funzione di HASHING:

Distribuire in modo UNIFORME e CASUALE i valori chiave nei bucket.

N.B.: è pesante cambiare la funzione di hashing dopo che la struttura d'accesso è stata riempita.

Esempio



File sequenziale ordinato (dati)

A	102	Rossi	120
B	110	Rossi	345
B	98	Bianchi	1000
E	17	Neri	1200
E	102	Verdi	950
F	113	Bianchi	4000
H	53	Neri	3000

Funzione di hashing

Filiale	h(f())
A	3
B	4
E	6
F	4
H	3

Strutture ad accesso calcolato (hashing)

Operazioni

RICERCA

- Dato un valore di chiave K trovare la corrispondente tupla
  - Calcolare b=h(f(K)) (costo zero)
  - Accedere al bucket b (costo: 1 accesso a pagina)
  - Accedere alle n tuple attraverso i puntatori del bucket (costo: m accessi a pagina con m<n)

INSERIMENTO E CANCELLAZIONE

- Di complessità simile alla ricerca.

Strutture ad accesso calcolato (hashing)

Osservazione

- La struttura ad accesso calcolato funziona se i buckets conservano un basso coefficiente di riempimento. Infatti il problema delle strutture ad accesso calcolato è la gestione delle COLLISIONI.
- COLLISIONE: si verifica quando, dati due valori di chiave K_1 e K_2 con $K_1 \neq K_2$, risulta:

$$h(f(K_1)) = h(f(K_2))$$

Un numero eccessivo di collisioni porta alla saturazione del bucket corrispondente.

Collisioni

Probabilità che un bucket riceva t chiavi su n inserimenti:

$$p(t) = \binom{n}{t}\left(\frac{1}{B}\right)^{t}\left(1-\frac{1}{B}\right)^{(n-t)}$$

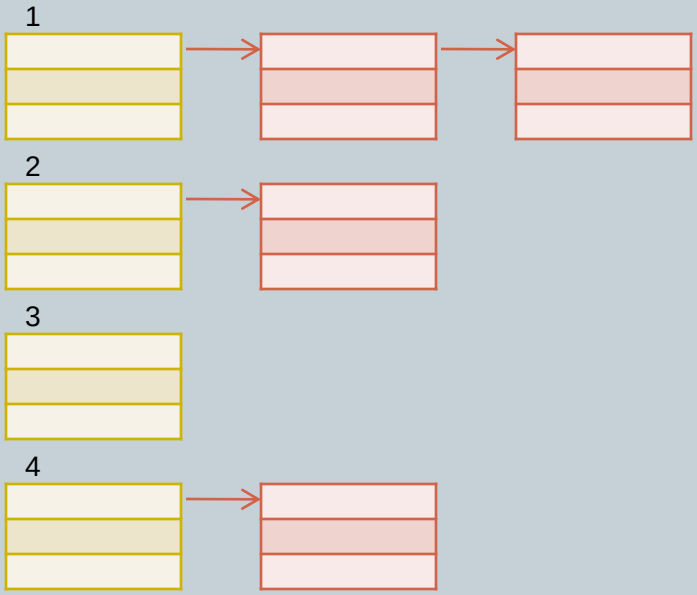
dove B è il numero totale di buckets.

Probabilità di avere più di F collisioni (F: numero di puntatori nel bucket):

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

Gestione delle Collisioni

Un numero eccessivo di collisioni sullo stesso indirizzo porta alla saturazione del bucket corrispondente. Per gestire tale situazione si prevede la possibilità di allocare Bucket di overflow, collegati al Bucket di base.



N.B.:
Le prestazioni della ricerca peggiorano in quanto individuato il bucket di base, potrebbe essere poi necessario accedere ai buckets di overflow.

Confronto B^+-tree e Hashing

Ricerca:

- Selezioni basate su condizioni di uguaglianza
  A = cost
  - **Hashing** (senza overflow buckets): tempo costante
  - B^+-tree: tempo logaritmico nel numero di chiavi
- Selezioni basate su intervalli (range)
  $A>cost_1$ AND $A<cost_2$
  - Hashing: numero elevato di selezioni su condizioni di uguaglianza per scandire tutti i valori del range
  - **B^+-tree**: tempo logaritmico per accedere al primo valore dell'intervallo, scansione sui nodi foglia fino all'ultimo valore compreso nel range.