Basi di Dati
Modulo Tecnologie

Strutture fisiche e strutture di accesso ai dati (III parte)

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 memoria secondaria**.
- 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

B+-tree: caratteristiche

- Caratteristiche generali dell'indice B+-tree:
  - È una struttura ad albero;
  - Ogni nodo viene memorizzato in 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 ordinati di chiave di ricerca e fino a n puntatori.

p_1	K_1	p_2	...	p_{m-1}	K_{m-1}	p_m

$m \leq n$

$i<j \Rightarrow K_i<K_j$

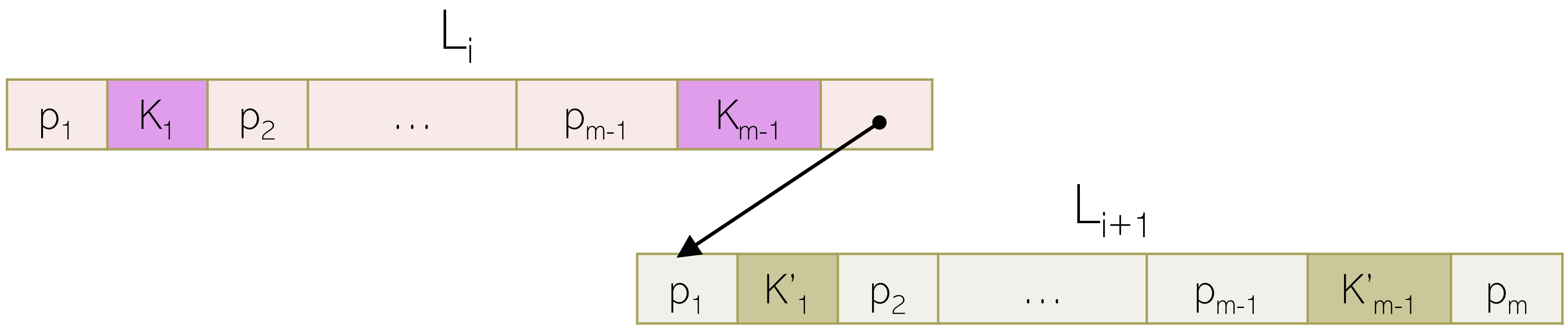
Punta alla prima tupla con chiave K_1 (indice primario).

Punta al bucket di puntatori verso le tuple con chiave K_1 (indice secondario).

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

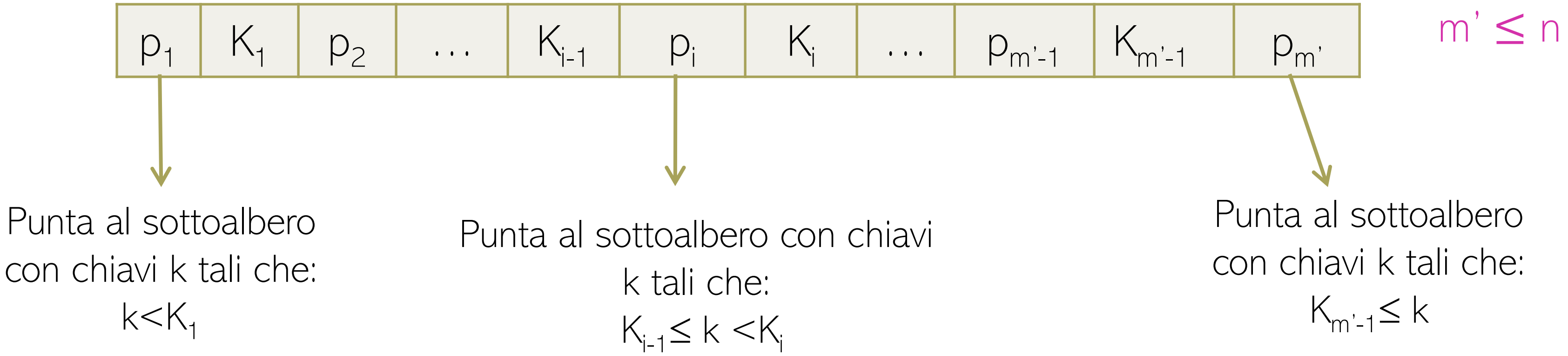
B+-tree: struttura

- 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
  - Sequenza di $m' \leq n$ valori ordinati di chiave
    - può contenere fino a n puntatori a nodo
  - Ogni chiave K_i è seguita da un puntatore p_i



B+-tree: vincoli di riempimento

- NODO FOGLIA (vincolo di riempimento con fan-out = n)
  - Ogni nodo foglia contiene un numero di valori chiave (#chiavi) vincolato come segue:

Arrotondamento all'intero superiore più vicino →

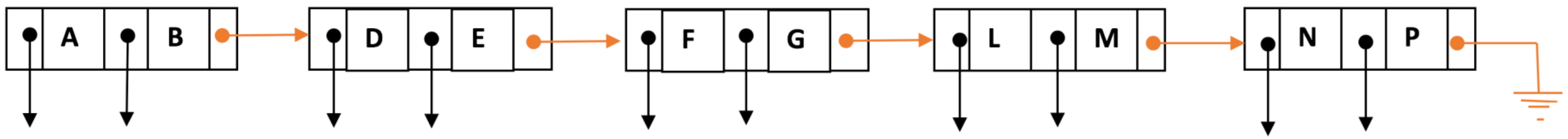
$$\lceil (n-1)/2 \rceil \leq \#chiavi \leq (n-1)$$

- NODO INTERMEDIO (vincolo di riempimento con fan-out = n)
  - Ogni nodo intermedio contiene un numero di puntatori (#puntatori) vincolato come segue (per la radice non vale il minimo):

Arrotondamento all'intero superiore più vicino →

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

Esempio di B+-tree

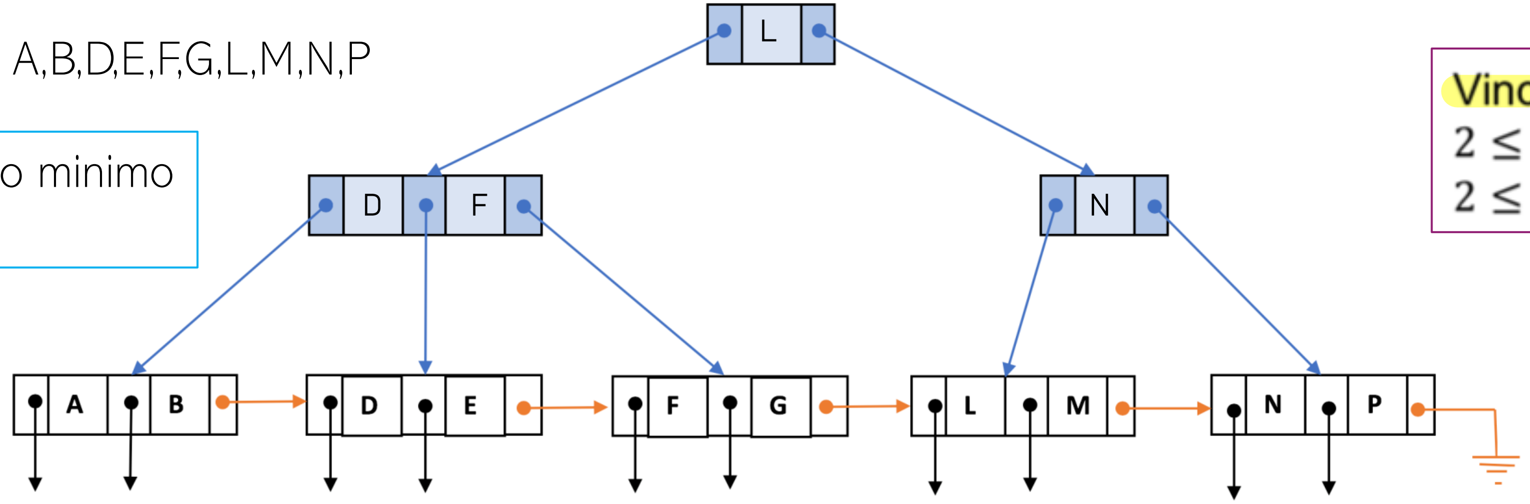


TUPLE della struttura fisica (indice primario) o BUCKETS puntatori (indice secondario)

Fan-out = 4

Valori chiave presenti: A,B,D,E,F,G,L,M,N,P

CASO A – riempimento minimo NODI FOGLIA



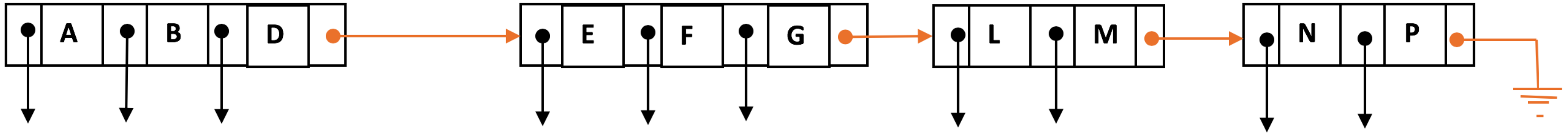
TUPLE della struttura fisica (indice primario) o BUCKETS puntatori (indice secondario)

Vincoli di riempimento:

$2 \leq \#chiavi \leq 3$

$2 \leq \#puntatori \leq 4$

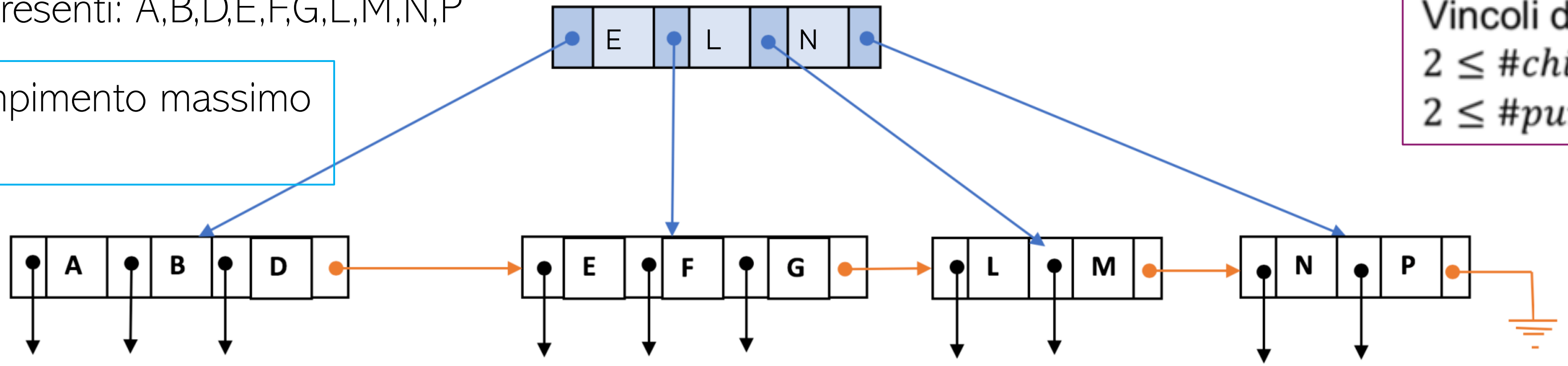
Esempio di B+-tree



TUPLE della struttura fisica (indice primario) o BUCKETS puntatori (indice secondario)

Fan-out = 4
Valori chiave presenti: A,B,D,E,F,G,L,M,N,P

CASO B – riempimento massimo
NODI FOGLIA



Vincoli di riempimento:
$2 \leq \#chiavi \leq 3$
$2 \leq \#puntatori \leq 4$

TUPLE della struttura fisica (indice primario) o BUCKETS puntatori (indice secondario)

B+-tree: Operazioni – Ricerca 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.

K = 15

p_1	3	p_2	...	15	p_i	19	...	p_{m-1}	123	p_m

Punta al sottoalbero con chiavi k tali che: $15 \leq k < 19$

- Se tale valore non esiste seguire il puntatore p_m.

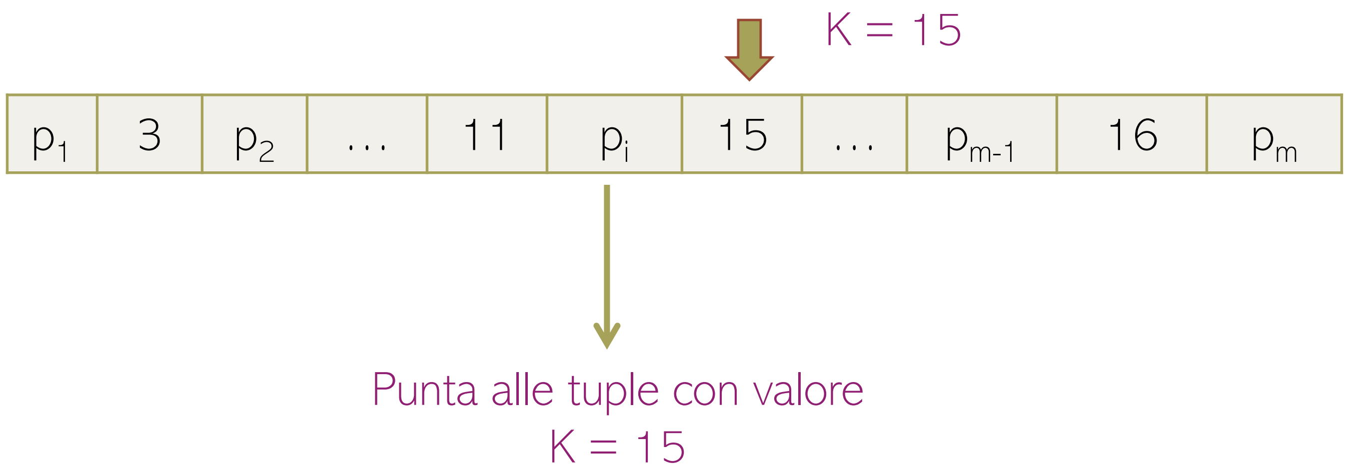
K = 124

p_1	3	p_2	...	15	p_i	19	...	p_{m-1}	123	p_m

Punta al sottoalbero con chiavi k tali che: $123 \leq k$

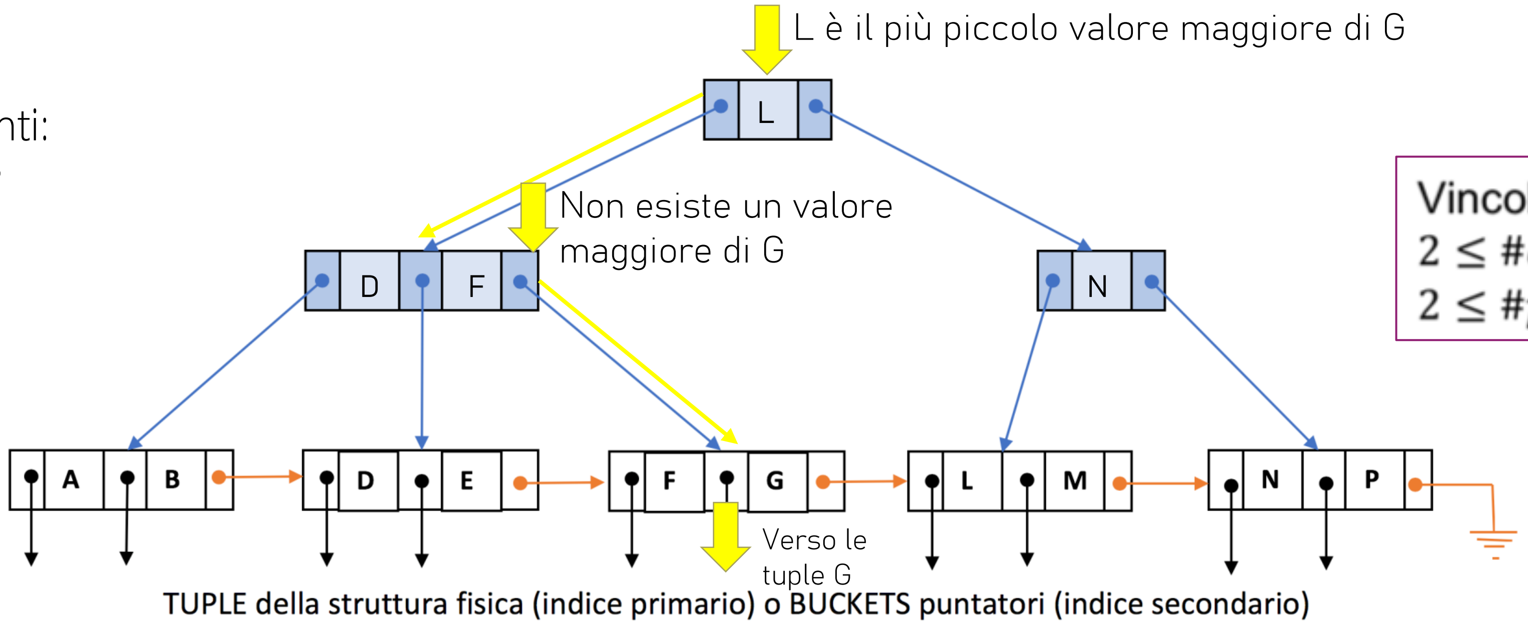
B+-tree: Operazioni – Ricerca con chiave K

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.



Esempio di ricerca nel B+-tree

Fan-out = 4
Valori chiave presenti:
A,B,D,E,F,G,L,M,N,P



Vincoli di riempimento:
$2 \leq \#chiavi \leq 3$
$2 \leq \#puntatori \leq 4$

Ricerca delle tuple con valore G della chiave di ricerca.
COME SI PROCEDE?

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: profondità dell'albero

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

Dimostrazione

- Dato un certo numero di valori chiave da inserire nell'albero (#valoriChiave) il numero massimo di nodi foglia è pari a:

$$NF\max = \frac{\#valoriChiave}{\lceil (n-1)/2 \rceil}$$

riempimento minimo

- Quindi partendo dal numero massimo di nodi foglia NF_{max} il numero massimo di livelli dell'albero, in presenza di nodi intermedi a riempimento minimo, risulta pari a:

$$NL_{max} = \log_{[n/2]}(NF_{max})$$

- Contando il livello dei nodi foglia si ottiene la profondità massima pari a: $1 + NL_{max}$

B+-tree: Operazioni – Inserimento con chiave K

- Passo 1 – ricerca del nodo foglia 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.

B+-tree: Operazioni – Inserimento con chiave K

K = 15

NF	p_1	3	p_2	...	8	p_i	12	...	p_{m-1}	16	p_m

NF	p_1	3	p_2	...	8	p_i	12	p_{i+1}	15	p_{m-1}	16	p_m

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

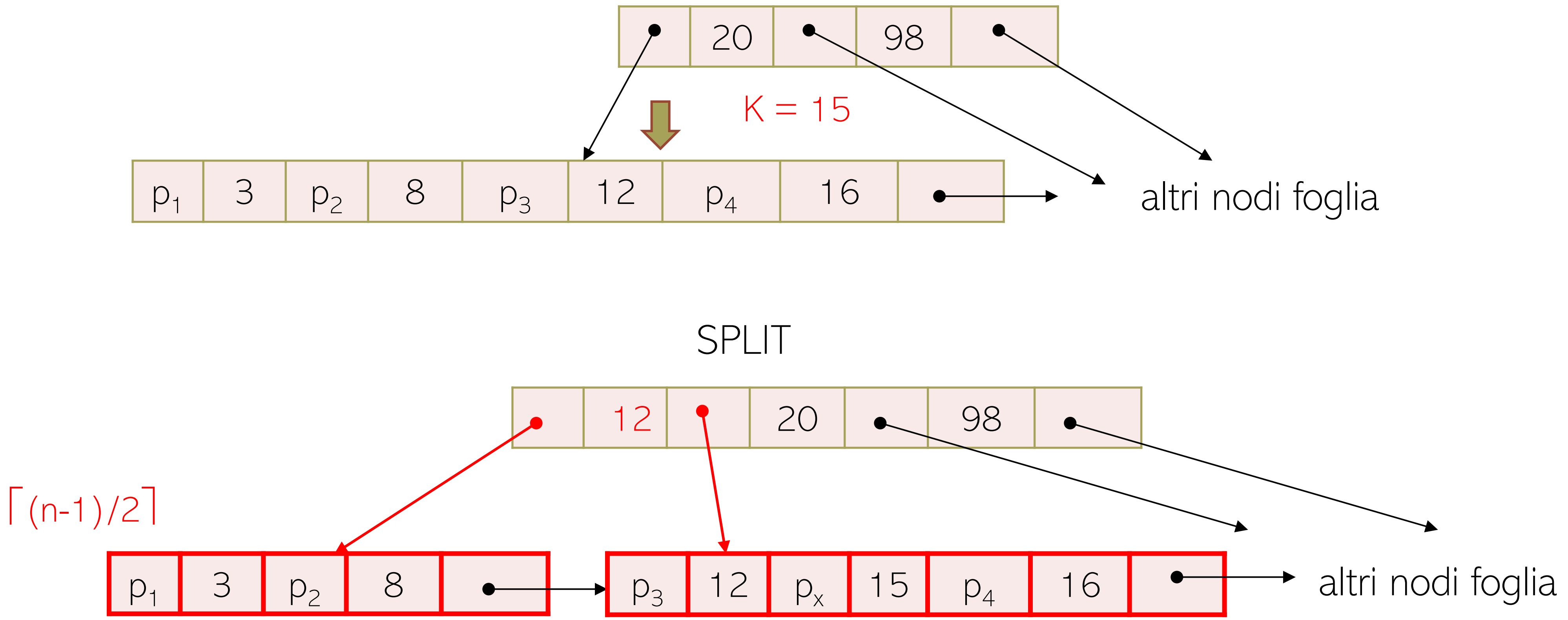
B+-tree: Operazioni – Inserimento con chiave K

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 con chiave K

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



B+-tree: Operazioni – Cancellazione con 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 ulteriore azione
  - Indice secondario: liberare il bucket di puntatori

B+-tree: Operazioni – Cancellazione con chiave K

K = 12

p_1	3	p_2	8	p_3	12	...	p_{m-1}	16	p_m

p_1	3	p_2	8	...	p_{m-1}	16	p_m

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

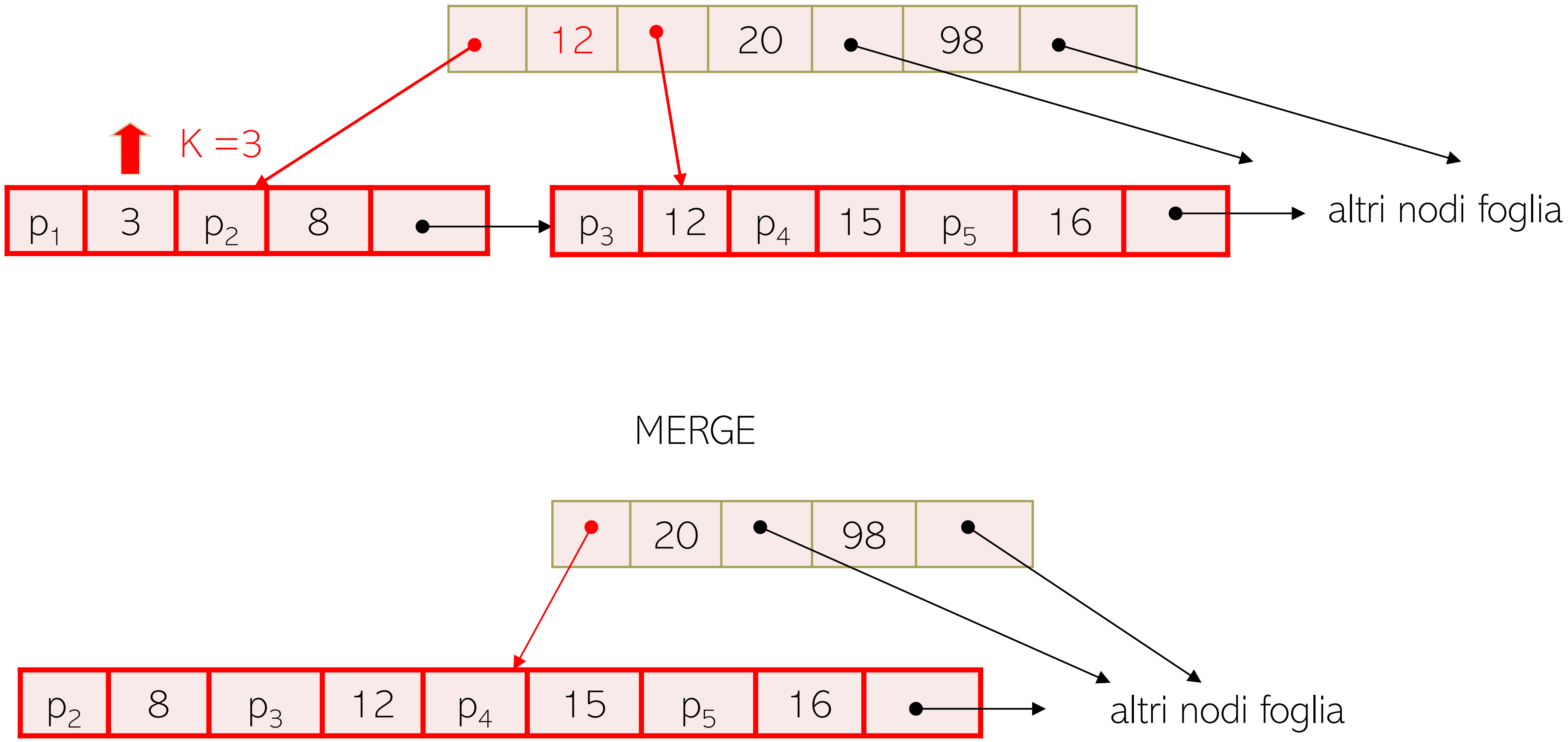
B+-tree: Operazioni – Cancellazione con chiave K

MERGE di un nodo foglia

- Se 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 $\lceil n/2\rceil$ puntatori presenti), il MERGE si propaga al padre e così via, se necessario, fino alla radice.

B+-tree: Operazioni – Cancellazione con chiave K

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



Hash

Strutture ad accesso calcolato (hashing)

- Una struttura ad accesso calcolato (hash) garantisce un accesso *associativo* ai dati → la locazione fisica dei dati dipende dal valore assunto da un campo chiave.
- Usare l'idea di array quando questa non è direttamente applicabile: usare un indice per l'accesso senza sprecare spazio.
- Si basano su una funzione di hash che mappa i valori della chiave di ricerca sugli indirizzi di memorizzazione delle tuple nelle pagine dati della memoria secondaria;
    - $h: K \rightarrow B$ K: dominio delle chiavi, B: dominio degli indirizzi
- Problema delle strutture ad accesso calcolato: sono sempre possibili delle collisioni, cioè valori di chiave di ricerca diversi, portano allo stesso valore di indice.
    - Una funzione di hash è buona, se minimizza la probabilità di collisioni multiple.

Strutture ad accesso calcolato (hashing)

- Si sfrutta le caratteristiche della memoria secondaria:
  - Il costo unitario delle operazioni è in numero di accessi ai blocchi, ciascuno dei quali può contenere diversi record.
  - *Fattore di riempimento*: frazione di spazio fisico mediamente utilizzata in ciascun blocco.
- Uso pratico di una funzione di hash negli indici:
  - Se *T* è il numero di tuple previsto nel file, *F* è il fattore di blocco (quante tuple per blocco) e *f* è il fattore di riemplimento: il file può prevedere un numero di blocchi B pari a $B = \lceil T/(f \times F) \rceil$
  - 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 della chiave nei bucket.

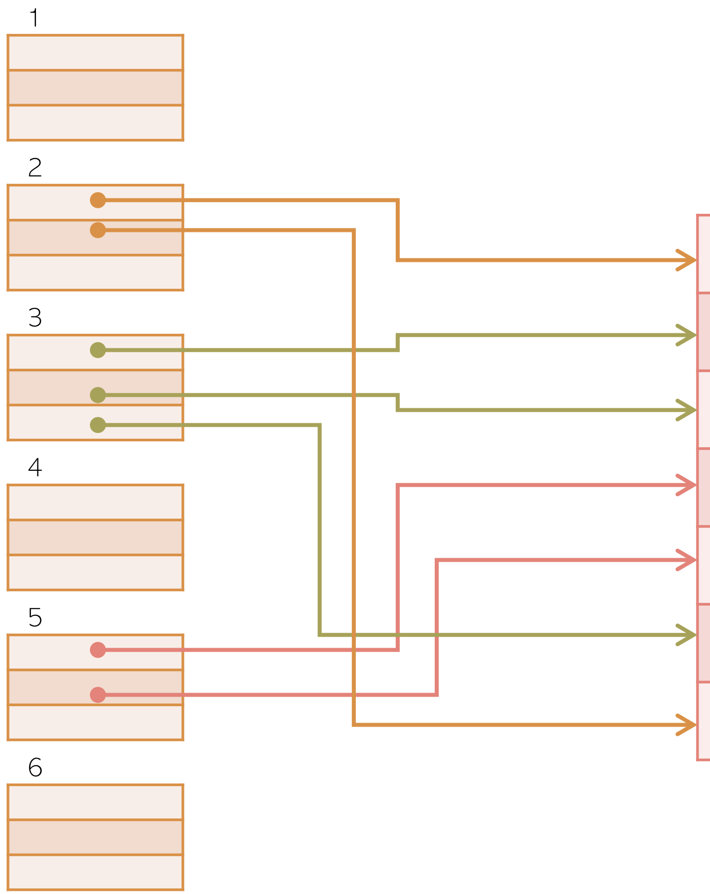
- N.B.: è pesante cambiare la funzione di hashing dopo che la struttura d'accesso è stata riempita; si deve ricostruire l'indice da capo.

Hashing: Esempio

1

$T = 7$
$F = 3$
$f = 0.4$ ➡ $B = 6$

Buckets



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	2
B	3
E	5
F	3
H	2

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 K1 e K2 con $K1 \neq K2$, risulta:

$$h(f(K1)) = h(f(K2))$$

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

Hashing: Collisioni

- Probabilità che uno stesso bucket riceva t chiavi su n inserimenti:

Coefficiente binomiale
Dato un insieme A di cardinalità n, il numero di sottoinsiemi di A di cardinalità t ≤ n, cioè il numero di combinazioni di n elementi presi t a t

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

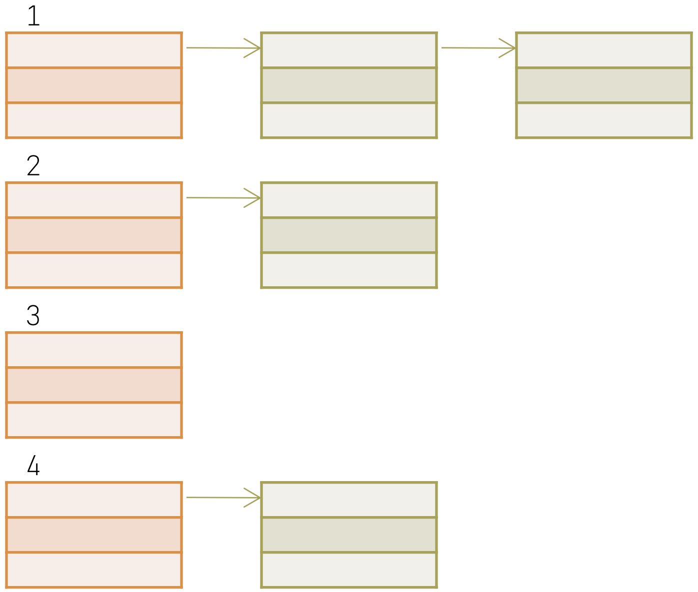
- dove B è il numero totale di buckets.

- Probabilità di avere più di F collisioni (F: fattore di blocco = numero di puntatori nel bucket):

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

Hashing: 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>cost1 AND A<cost2
  - 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 dei nodi foglia (grazie all'ultimo puntatore) fino all'ultimo valore compreso nel range.

Confronto B+-tree e Hashing

Inserimenti e cancellazioni:

- Hashing: tempo costante + gestione overflow
- B+-tree: tempo logaritmico nel numero di chiavi + split/merge