

METODI DI ACCESSO (INDICI)

METODI DI ACCESSO

(INDICI B⁺-Tree)

Nelle lezioni precedenti

- **Avete visto:**
 - le caratteristiche delle principali unità di **memoria permanente**
 - la **struttura** dei files
 - il loro **ordinamento**

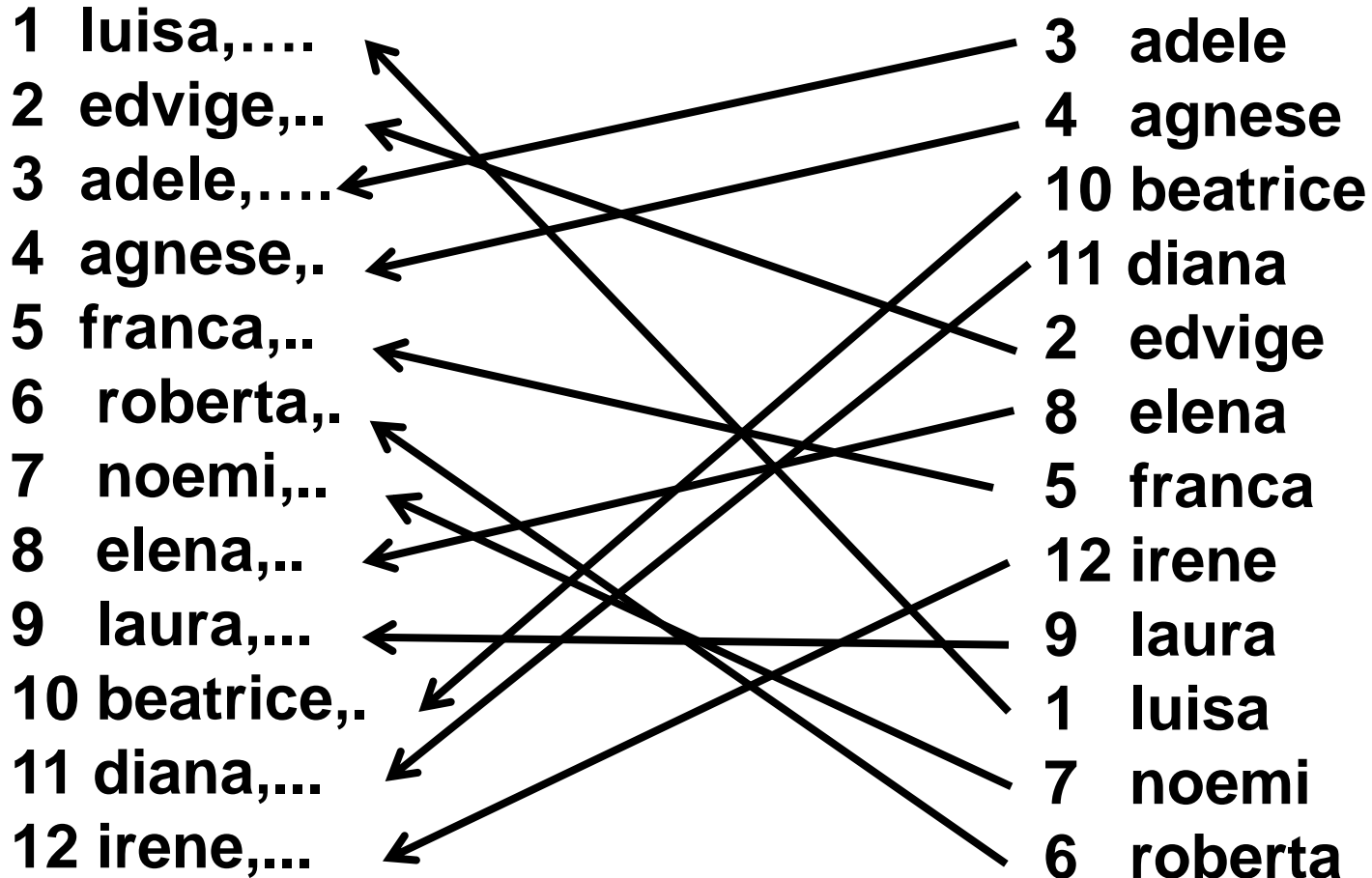
In questa lezione

- **Presenteremo:**
 - le caratteristiche principali degli **indici** per l'accesso alle tuple (record)

organizzazioni con indice

- **IDEA DI BASE** : associare ad un file una tabella nella quale l'entrata i -esima memorizza una coppia del tipo (k_i, p_i) dove :
 - k_i è un valore di chiave
 - p_i è un riferimento al record (tupla) con valore di chiave k_i
 - l'indice è **ordinato**: le coppie (k_i, p_i) sono ordinate in base ai valori di k

organizzazioni con indice

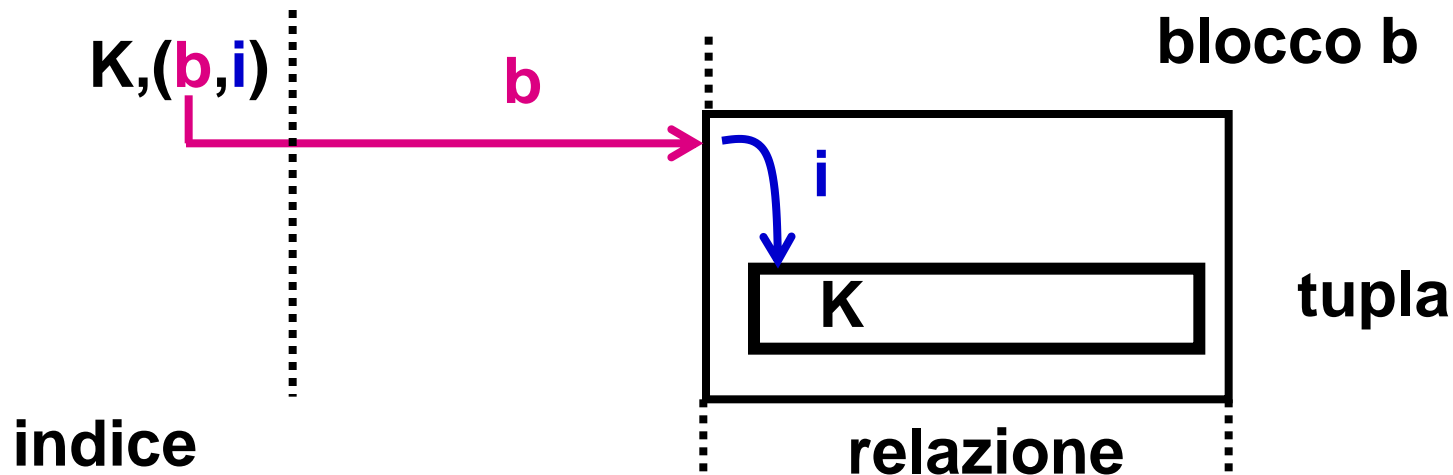


organizzazioni con indice

- Nell'esempio abbiamo mostrato solamente la colonna chiave e la posizione della tupla, mentre l'indice è un file costituito da **mini-tuple** ed occupa generalmente uno spazio di memoria di circa 5÷20% della relazione
- **ordinare** il file indice è quindi più veloce
- l'indice può essere **costruito su attributi qualsiasi** (su “chiave” di ricerca, non necessariamente chiave nel senso relazionale)
- l'indice fornisce una **visione ordinata** della relazione anche su colonne non di ordinamento

organizzazioni con indice

- I riferimenti (4÷8 byte) vengono denominati: indirizzi, puntatori o, più comunemente, **TID** (tuple identifier) o **row-id** (row identifier)
- sono costituiti da una **coppia: id. di blocco (pagina), indirizzo nel blocco:**



L'indirizzo nel blocco è a sua volta indiretto

organizzazioni con indice

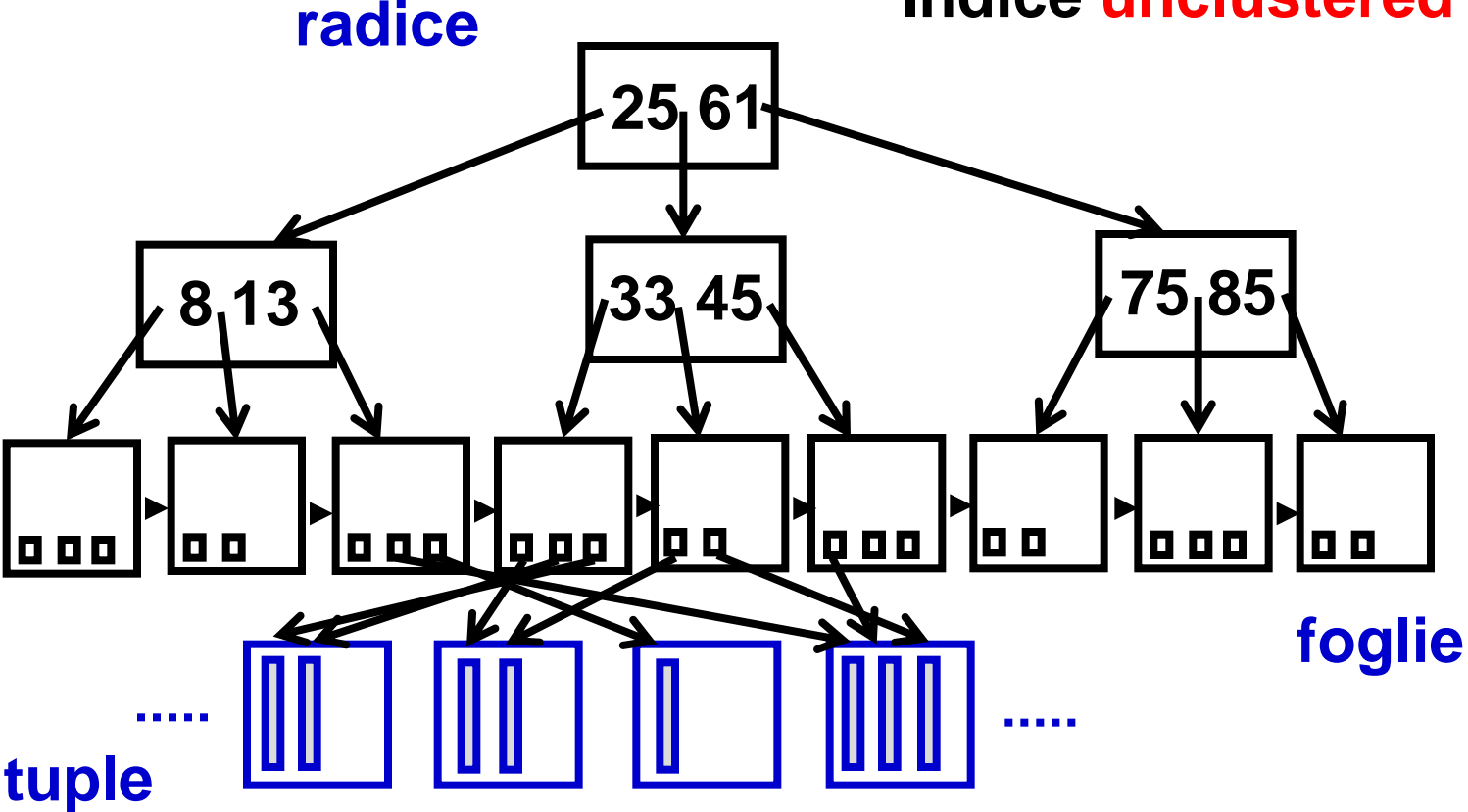
- **Metodo di accesso**
data una chiave K :
 1. accesso all'**indice**
 2. ricerca della **coppia** $(K,(b,i))$
 3. accesso al **blocco** dati b
 4. accesso alla **tupla** contenente K
(indirizzo i)
- le **coppie** $(K,(b,i))$ vengono **collezionate in blocchi ed ordinate**, questi blocchi si chiamano **FOGLIE** dell'indice e sono tra loro collegati da puntatori.

organizzazioni con indice

- Al di sopra delle foglie si costruisce **un'organizzazione ad albero** a più livelli di indirizzamento:
 - **ogni foglia** è rappresentata da **una coppia** (K,b) nel livello immediatamente superiore (K è il valore minore),
 - i blocchi del livello sopra le foglie sono a loro volta rappresentati a **livello superiore**, e così via fino ad avere un solo blocco chiamato **radice**

organizzazioni con indice

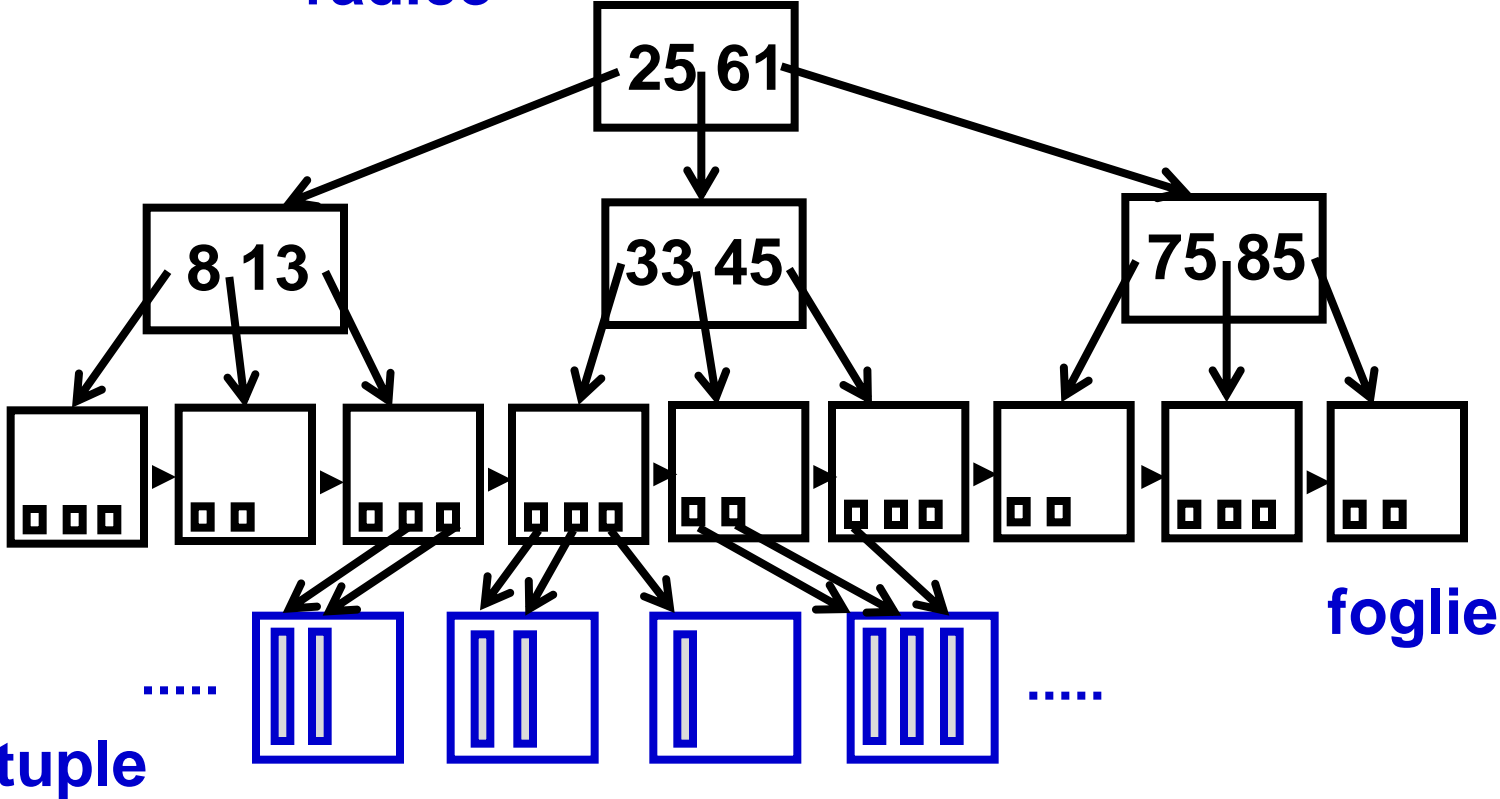
Indice **unclustered**



organizzazioni con indice

Indice **clustered**

radice

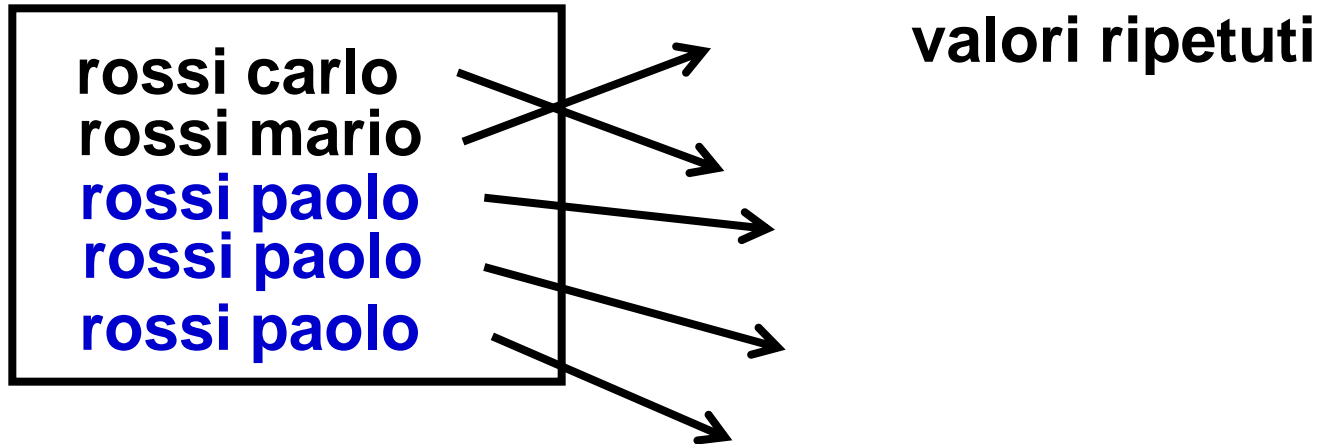


(non c'è "incrocio" di puntatori)

indici

- L'indice si dice **clustered** se è costruito su un attributo di ordinamento, altrimenti si dice **unclustered**
- su una relazione si possono costruire **un** indice clustered e **più di un** indice unclustered
- l'indice può essere anche **multiattributo** (multicolonna), ad es. K :<cognome,nome>
- nel caso di clausola **UNIQUE** l'indice garantisce la **non ripetizione dei valori**
- l'indice può essere costruito su attributi con **valori ripetuti**, in tal caso la coppia (K, p) diventa **(K, (p1, p2,, pk))**

indici

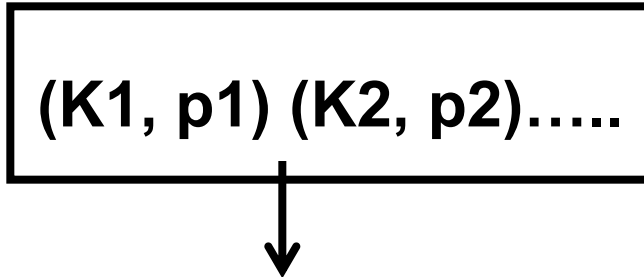


eliminazione dei
valori ripetuti e
riduzione di spazio

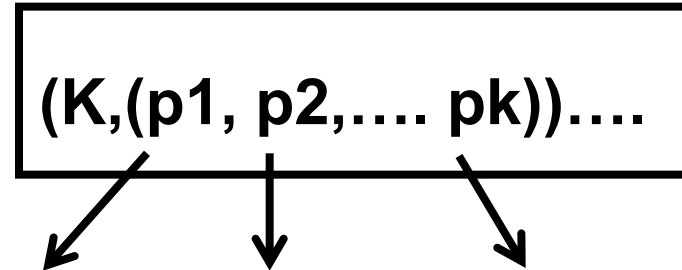


indici

unique



generico



In SQL:

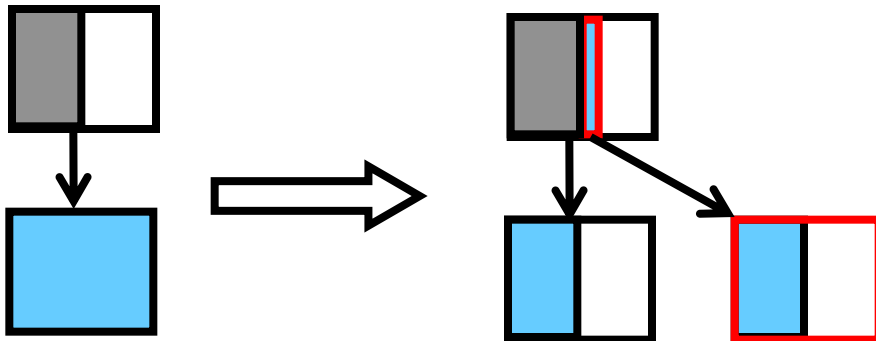
- **CREATE {UNIQUE | CLUSTER} INDEX** nome-indice **ON** nome-relazione (nome-attributo { **ASC | DESC** }, ...)
- **DROP INDEX** nome-indice
- **ALTER INDEX** nome-indice **TO CLUSTER**

indici B⁺ tree

- **B⁺ tree**: l'albero è associato ad un SW di gestione che lo mantiene **sempre equilibrato**: ogni percorso dalla radice ad una foglia ha sempre la stessa **lunghezza h**, chiamata **altezza** del B⁺ tree
- ogni blocco (nodo) intermedio ha sempre almeno **O+1** e non più di **2×O+1** figli, contiene almeno **O** e non più di **2×O** valori dell'attributo
- la radice può avere da **2** a non più di **2×O +1** figli, contiene da **1** a non più di **2×O** valori
- **O** si chiama **ordine** del B⁺ tree
- Le foglie sono riempite dal **50 al 100 %**

indici B⁺ tree

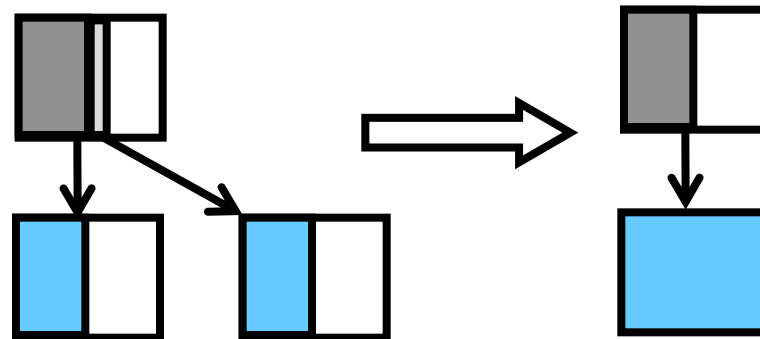
inserimento



block split
(si può propagare fino alla radice)

eliminazione

underflow
(si può propagare fino alla radice)



prestazioni del B⁺ tree

Facciamo riferimento a:

- dimensione del blocco **D** (es. 4096)
- dimensione **L(tid)** del **TID** e del puntatore **L(p)** ai nodi intermedi (es. 4 per entrambi)
- dimensione **L(k)** del valore **K**

Calcoliamo :

- l'ordine **O**
- il numero di foglie **NF**
- l'altezza **H**

prestazioni del B⁺ tree

Ordine O:

consideriamo il vincolo

$$2 \times O \times L(k) + (2 \times O + 1) \times L(p) \leq D$$

da cui

$$O = \lfloor (D - L(p)) / (2 \times (L(k) + L(p))) \rfloor$$

- con una dimensione L(k) di 10 si ha
O = 146 e 4 byte inutilizzati per nodo
- con una dimensione L(k) di 40 si ha
O = 46 e 44 byte inutilizzati per nodo

prestazioni del B⁺ tree

Numero di foglie NF:

- dipende dal numero di tuple **NT** e dal **fattore di riempimento medio** dei blocchi = $\ln 2 \approx 69\%$
- ricordiamo che nelle foglie si avranno un numero di **TID = NT** ed un numero di valori **NK** da cui

$$NF = \lceil (NK \times L(k) + NT \times L(p)) / (D \times \ln 2) \rceil$$

- con una dimensione **L(k) = 10**, **NT = 10⁶**,
NK = 10⁴ si ha: **NF = 5776**

prestazioni del B⁺ tree

Altezza minima:

- l'altezza minima **Hmin** si può ottenere quando **tutti** i nodi sono **pieni**, poiché al **penultimo livello** sono necessari **NF** puntatori alle foglie, si ha:

$$NF \leq (2 \times O + 1)^{Hmin-1} \quad \text{e quindi}$$

$$Hmin = 1 + \lceil \text{Log}_{(2 \times O + 1)} NF \rceil$$

prestazioni del B⁺ tree

Altezza massima:

- l'altezza massima **Hmax** si può ottenere quando tutti i nodi sono pieni **per metà** e la radice contiene **due soli** puntatori, poiché al **penultimo livello** sono necessari **NF** puntatori alle foglie, si ha:

$$NF \leq 2 \times (O + 1)^{Hmax-2} \quad \text{e quindi}$$

$$Hmax = 2 + \lfloor \text{Log}_{(O + 1)} (NF / 2) \rfloor$$

prestazioni del B⁺ tree

- Esempio

$D = 1024$, $L(\text{tid}) = 4$, $NT = 10^6$

NK	$L(k) = 10$			$L(k) = 20$		
	NF	Hmax	Hmin	NF	Hmax	Hmin
10^3	5650	4	4	5664	4	4
10^4	5776	4	4	5918	4	4
10^5	7045	4	4	8454	4	4
10^6	19725	4	4	33814	5	4

si tratta in generale di alberi abbastanza **bassi**

prestazioni del B⁺ tree

- Continuando l'esempio precedente, con NB= 10^5 , 10^6 tuple, 10 per blocco L(k) = 10
 - $C_{seq} = 5 \times 10^4$
 - $C_{sort} = 1.2 \times 10^6$ e $C_{bin} = 17$
 - $C_{ind} = 4 + 1 = 5$,
bisogna sommare l'accesso al blocco che contiene la tupla cercata (+1)
- si può anche ipotizzare che se le richieste sono molte, durante l'esercizio, la radice ed i livelli superiori siano già in memoria:
 - $C_{ind} = 1 + 1 = 2$

prestazioni del B⁺ tree

- costo di costruzione dell'indice:
- lettura della relazione,
- estrazione delle coppie, costruzione delle foglie
- sort/merge delle stesse

$$-C_{\text{indice}} = NB + NF + 2 \times NF \times (PM+1)$$

continuando l'esempio precedente, con M e NM = 8 ed inglobando il passo di sort nella estrazione delle coppie:

$$C_{\text{indice}} = NB + NF + 2 \times NF \times PM$$

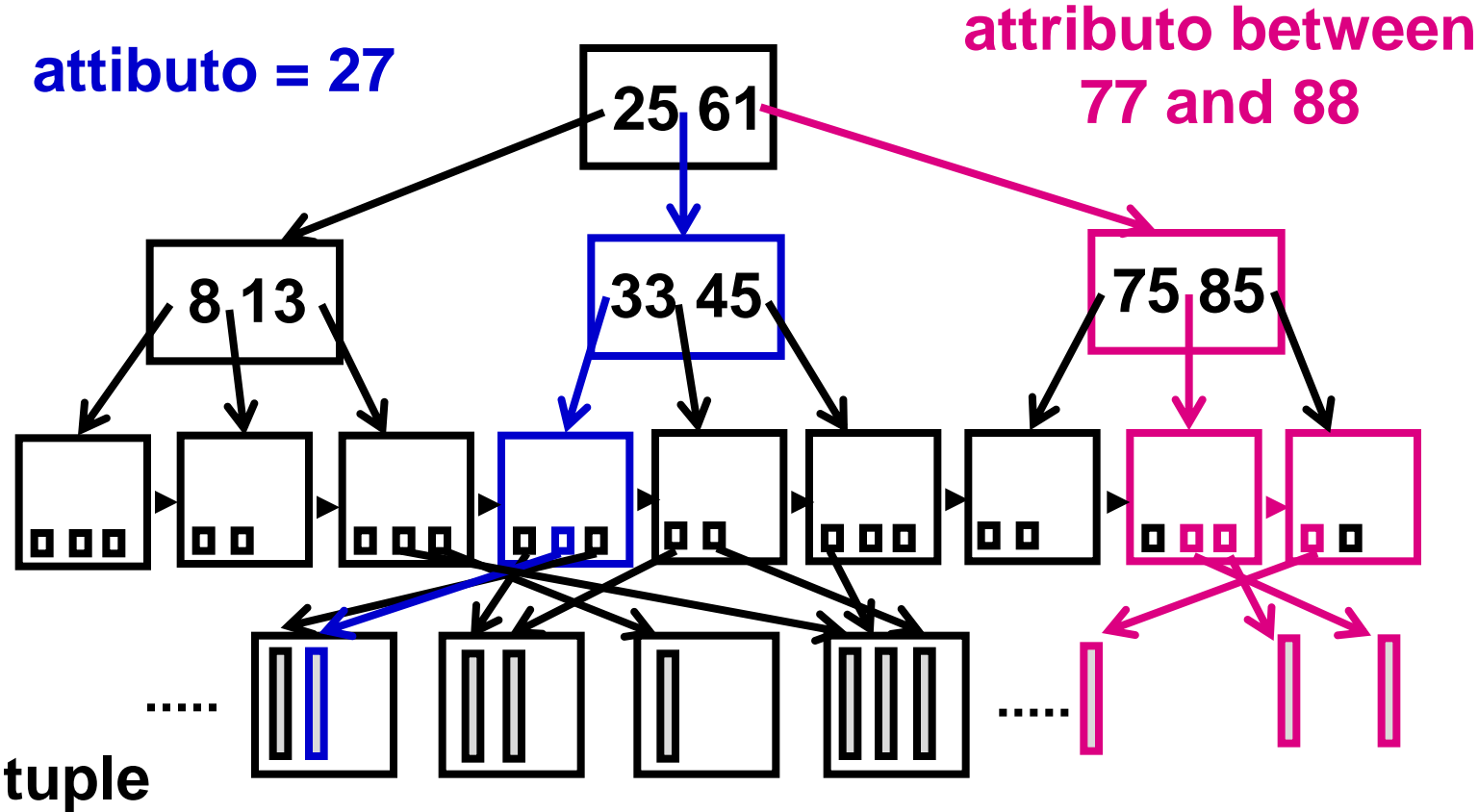
$$C_{\text{indice}} = 10^5 + 1.98 \times 10^5 \approx 3 \times 10^5$$

prestazioni del B⁺ tree

Utilizzo dell'indice:

- accesso veloce alla tupla per K **unique**
- accesso veloce alle tuple per K **replicata**
- accesso veloce alle tuple per **range** di K
(K < valore, K BETWEEN k1 AND k2)
- facilita **ORDER BY** e **GROUP BY**

organizzazioni con indice



Ricerca di un valore di chiave K

```
/* NODO: (Q0, K1, Q1, K2, Q2, ..., Kn, Qn) */
```

```
Q := root;
```

```
while ( Q is not a leaf node ) do
```

```
{ if K < K1 then Q := Q0
```

```
  else if (∃i: Ki ≤ K < Ki+1) then Q = Qi
```

```
    else Q := Qn
```

```
};
```

```
If (∃i: K = Ki) then
```

```
  for each TID in Qi do FetchTuple(TID);
```

Ricerca di un range [K1,K2]

```
/* FOGLIA: (Qprev, K1, (TID1,...,TIDp), ...,  
Kn, (TIDr,..., TIDs), Qnext) */
```

```
Qi := FindKey(K1);
```

```
while ( Ki<=K2 ) do {
```

```
    for each TID in Qi do FetchTuple(TID);
```

```
    if ( i<n ) then ++i else Qi := Qnext
```

```
}
```

compressione della chiave

- **FORWARD COMPRESSION**

- Si registrano: il numero di caratteri uguali ed i caratteri finali che differenziano dal valore precedente.

- Es:

JOHNSON

JOHNSTON

JOHNSTOWN

JOHNTON

0	JOHNSON
5	TON
7	WN
4	TON

- **REAR COMPRESSION** (Riporta i caratteri iniziali) è inefficace

compressione della chiave

- **FORWARD / REAR CON SOPPRESSIONE**

Si segue la regola della Forward e si sopprimono dei caratteri che non servono a differenziare ciascun valore dal precedente e dal successivo:

0 JOHNSON

5 TON

7 WN

4 TON



0 JOHNS
5 TO
7 W
4 T

JOHNS ?????

JOHNSTO ??

JOHNSTOW ?

JOHNT ?????

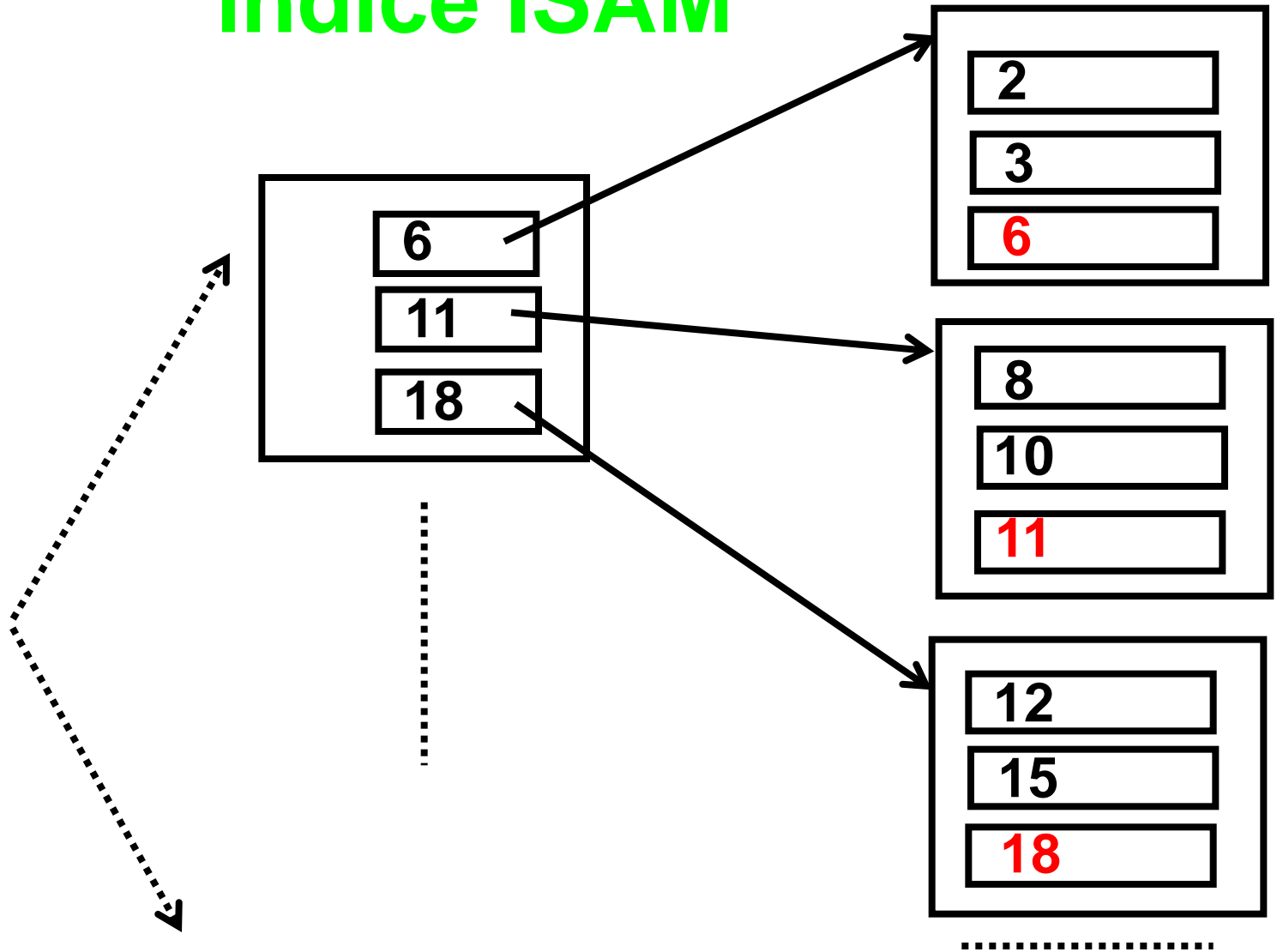
che corrisponde a: 

si perdono quindi i valori completi della chiave

indice ISAM

- **ISAM: Indexed Sequential Access Method**
 - presuppone il sort del file sulla colonna dove è costruito ISAM
 - a livello foglie riporta solo il valore di chiave più grande presente in ciascun blocco dati ed il relativo puntatore al blocco: a differenza del B⁺tree contiene NB coppie (chiave, id) invece di NT
 - è statico: se un valore viene cancellato o inserito l'indice non si modifica, presuppone l'uso di overflow nel file indirizzato, va ricostruito a seguito di molti inserimenti.
 - è generalmente più basso del B⁺.

indice ISAM



indice ISAM

- ISAM: Indexed Sequential Access Method

Numero di foglie NF:

- dipende dal numero di blocchi **NB** e dal **fattore di riempimento** dei blocchi **fr** (es.= **100 %**)
- ricordiamo che nelle foglie si avranno un numero di **ID = NB** ed un numero di valori **NB** da cui

$$NF = \lceil (NB \times L(k) + NB \times L(p)) / (D \times fr) \rceil$$

con **D = 1024**, **L(k) = 10**, **L(p) = 4** e **NB = 10⁵** si ha: **NF = 5833**

- per l'altezza calcolabile come il caso minimo del B⁺ si ha **4**

nelle prossime lezioni

- **L'uso degli indici per l'accesso**
- **la stima del costo delle interrogazioni SQL**