Architetture dei sistemi di gestione dati

1

Contenuti del corso

- Organizzazione dei dati su memoria secondaria
  - strutture di memorizzazione
  - strutture di indicizzazione (per dati strutturati e non)
- Strategie di elaborazione di interrogazioni
- Controllo della concorrenza
- Gestione del ripristino
- Autorizzazione e controllo degli accessi

2

Contenuti del corso

- Architetture distribuite e multidatabase
  - basi di dati distribuite
  - transazioni in ambiente distribuito
  - sistemi multidatabase e interoperabilità
- Basi di dati e Web
  - accesso a basi di dati attraverso il Web
  - dati semistrutturati e XML
  - gestione di documenti XML mediante DBMS

3

Architettura di un DBMS

4

Architettura di un DBMS

- Un DBMS deve garantire una gestione dei dati:
  - efficiente
  - concorrente
  - affidabile
  - integra
  - sicura (protetta)
- Ciascuno degli aspetti precedenti è supportato dal DBMS da specifiche componenti, che complessivamente rappresentano l'architettura del sistema

5

Componenti di un DBMS

- **Efficienza**:
  - **File system**: gestisce l'allocazione dello spazio su disco e le strutture dati usate per rappresentare le informazioni memorizzate su disco
  - **Buffer manager**: responsabile del trasferimento delle informazioni tra disco e main memory
  - **Query parser**: traduce i comandi del DDL e del DML in un formato interno (parse tree)
  - **Optimizer**: trasforma una richiesta utente in una equivalente ma più efficiente

6

Componenti di un DBMS

- **Affidabilità**:
  - **Recovery manager**: assicura che il DB rimanga in uno stato consistente a fronte di cadute del sistema
- **Concorrenza**:
  - **Concurrency controller**: assicura che interazioni concorrenti procedano senza conflitti
- **Integrità**:
  - **Integrity manager**: controlla che i vincoli di integrità (per esempio le chiavi) siano verificati
- **Sicurezza**
  - **Authorization manager**: controlla che gli utenti abbiano i diritti di accesso ai dati

7

Componenti di un DBMS

- Un DBMS contiene inoltre alcune strutture dati che includono:
  - i file con i dati (cioè i file per memorizzare il DB stesso)
  - i file dei dati di sistema (che includono il dizionario dei dati e le autorizzazioni)
  - indici (esempio B-tree o tabelle hash)
  - dati statistici (esempio il numero di tuple in una relazione) che sono usati per determinare la strategia ottima di esecuzione

8

Efficienza

- Finora (Basi di dati 1) abbiamo visto modelli di DBMS ad alto livello (**livello logico**)
- Tale livello è il livello corretto per gli utenti del DB
- Un fattore importante nell'accettazione da parte dell'utente è dato dalle prestazioni
- Le prestazioni del DBMS dipendono dall'efficienza delle strutture dati e dall'efficienza del sistema nell'operare su tali strutture dati

9

Efficienza

- Esistono varie strutture alternative per implementare un modello dei dati
- La scelta delle strutture più efficienti dipende dal tipo di accessi che si eseguono sui dati
- Normalmente un DBMS ha le proprie strategie di implementazione di un modello dei dati
- Tuttavia l'utente (esperto) può influenzare le scelte fatte dal sistema (**livello fisico**)

10

Supporti di memorizzazione

- I dati memorizzati in una base di dati devono essere fisicamente memorizzati su un supporto fisico di memorizzazione
- Memoria primaria
  - memoria principale (main memory) e memorie più piccole e più veloci
  - i dati sono manipolati direttamente dalla CPU
  - accesso veloce ai dati
  - capacità di memorizzazione limitata
  - volatile (il contenuto è perso se va via la corrente o si ha una caduta di sistema)

11

Supporti di memorizzazione

- Memoria secondaria
  - Dischi magnetici, dischi ottici e nastri
  - capacità maggiore, costo inferiore e accesso più lento
  - necessità di trasferire i dati in memoria principale per elaborazione dalla CPU
  - non volatile
- Basi di dati in genere sono memorizzate su memoria secondaria (dischi magnetici)
  - troppo grosse per risiedere in memoria principale
  - maggiori garanzie di persistenza dei dati
  - costo per unità di memorizzazione decisamente inferiore

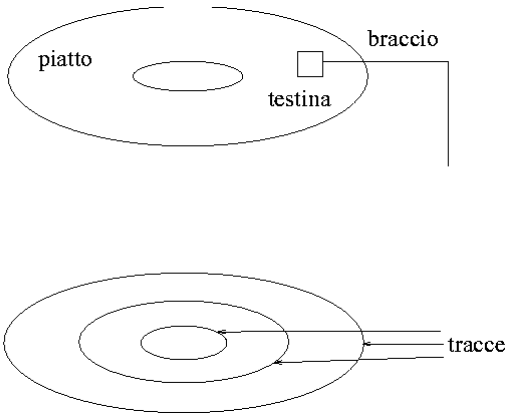
12

Dischi

- L'informazione è memorizzata su una superficie del disco in cerchi concentrici di piccola ampiezza, ognuno con un diametro distinto, detti **tracce**
- per i dischi a più piatti, le tracce con lo stesso diametro sulle varie superfici sono dette **cilindro**
- dati memorizzati su uno stesso cilindro possono essere recuperati molto più velocemente che non dati distribuiti su diversi cilindri
- il meccanismo hardware che legge e scrive è la testina, collegato ad un braccio meccanico

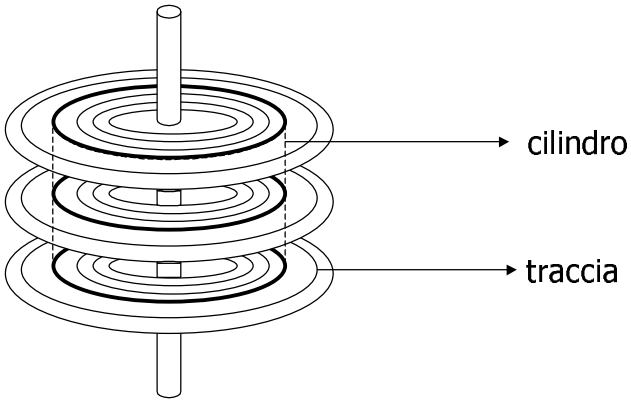
13

Dischi



14

Dischi a più piatti



15

Dischi

- nel caso di dischi a più piatti, tutti i bracci sono collegati ad un'unità detta **actuator**, che muove contemporaneamente tutte le testine posizionandole sul cilindro delle tracce specificate in un indirizzo
- **tempo di latenza del disco**: tempo necessario affinché tutta una traccia passi sotto al braccio (tempo molto breve)
- **tempo di seek**: tempo necessario per posizionare i bracci su un certo cilindro (tempo molto più alto rispetto al tempo di latenza)

16

Dischi

- I dati sono trasferiti tra il disco e la memoria principale in unità chiamate blocchi
  - un blocco è una sequenza di byte contigui memorizzati in una stessa traccia di un singolo cilindro
  - la dimensione del blocco dipende dal sistema operativo e varia tipicamente tra 512 e 4096 byte
  - il tempo di trasferimento di un blocco è il tempo impiegato dalla testina per trasferire un blocco nel buffer, una volta posizionata all'inizio del blocco
  - tale tempo è molto più breve del tempo necessario per posizionare la testina all'inizio del blocco (tempo di seek)

17

Organizzazione di file

- I dati sono generalmente memorizzati in forma di record
  - un record è costituito da un insieme di valori collegati
  - ogni valore è formato da uno o più byte e corrisponde ad un campo del record
- una collezione di nomi di campi a cui sono associati i tipi corrispondenti costituisce un *tipo di record*
- il numero di byte necessari per la memorizzazione di un valore di un certo tipo è fissato per ogni sistema

18

Organizzazione di file

un record del seguente tipo

Nome del Tipo di Record

```
type Impiegato = record
```

Nomi dei Campi Tipi dei Campi

```
Imp#: integer;
Nome: array[1..20] of char;
Mansione: array[1..10] of char;
Data A: date;
Stipendio: integer;
Premio P: integer;
Dip#: integer;
```

può essere memorizzato in 50 byte

(4*4=16 per gli interi + 4 per la data + 10 + 20 per le stringhe)

19

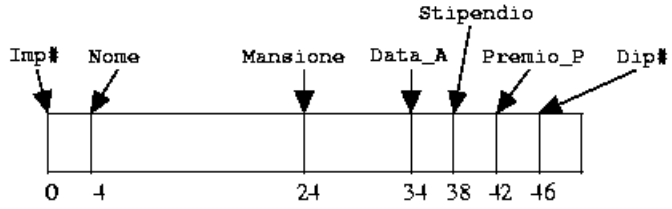
Organizzazione di file

- un file è una sequenza di record
- file con record a **lunghezza fissa** se tutti i record memorizzati nel file hanno la stessa dimensione (in byte)
- un file può contenere record a **lunghezza variabile** per varie ragioni:
  - campi di dimensione variabile
  - campi multivalore
  - campi opzionali
  - file eterogenei: il file contiene record di tipi differenti, e quindi di dimensioni differenti

    (informazioni collegate ma di tipi differenti memorizzate vicine sullo stesso blocco di disco)

20

File con record a lunghezza fissa



21

File con record a lunghezza fissa

Cancellazione di un record:

- inserzioni più frequenti delle cancellazioni
  - non si effettuano spostamenti di record e si lascia libero lo spazio occupato dal record cancellato, poi utilizzato alla prossima inserzione
- è necessario allocare delle strutture ausiliarie (file header) per determinare velocemente dove effettuare l'inserzione (il file header contiene il puntatore al primo record cancellato e i record cancellati sono collegati a lista)

22

File con record a lunghezza variabile

- tipo di record Impiegato
  - un attributo multivalore: la mansione può non essere unica
  - un attributo opzionale: il premio di produzione può non essere specificato
- sono possibili diverse rappresentazioni

23

File con record a lunghezza variabile

(a)

7369	Rossi	ingegnere	dirigente	17-Dic-80	1600	500	20
7499	Andrei	tecnico	20-Feb-81	800	30		
7654	Martini	segretaria	interprete	28-Set-81	800	30	
7782	Neri	ingegnere	01-Giu-81	2.450	200	30	

(b)

7369	Rossi	ingegnere	dirigente	17-Dic-80	1600	500	20
7499	Andrei	tecnico	#	20-Feb-81	800	#	30
7654	Martini	segretaria	interprete	28-Set-81	800	#	30
7782	Neri	ingegnere	#	01-Giu-81	2450	200	30

(c)

	7369	Rossi	ingegnere	17-Dic-80	1600	500	20
	7499	Andrei	tecnico	20-Feb-81	800	#	30
	7654	Martini	segretaria	28-Set-81	800	#	30
	7782	Neri	ingegnere	01-Giu-81	2450	200	30

dirigente
interprete

24

File con record a lunghezza variabile

- **Rappresentazione byte string**
  - si aggiunge un simbolo speciale end-of-record che indica la fine di ogni record
  - ogni record viene memorizzato come una sequenza di byte consecutivi
  - uso di speciali caratteri separatori, che non appaiono come valore in nessun campo, per terminare i campi a lunghezza variabile

25

File con record a lunghezza variabile

- **byte string - principali svantaggi:**
  - non è facile riutilizzare lo spazio occupato da un record cancellato; esistono tecniche per gestire cancellazioni ed inserzioni ma tendono a generare frammentazione
  - se un record aumenta di lunghezza deve essere spostato
    - questo può essere costoso se tale record è puntato

26

File con record a lunghezza variabile

- **rappresentazione fixed-length reserved-space**
  - si rappresenta un file con record a lunghezza variabile mediante file con record a lunghezza fissa
  - si riserva per i record lo spazio massimo occupabile
  - problema: spreco di spazio => lentezza nell'accesso (i record tenderanno a disperdersi su molti blocchi)

27

File con record a lunghezza variabile

- **rappresentazione fixed-length**
  - si rappresenta un record a lunghezza variabile con una lista di record a lunghezza fissa, collegati tramite puntatori
  - per evitare di dover ripetere i campi che assumono comunque un solo valore si usano due tipi di blocchi nel file:
    - **anchor block**, contenenti il primo record di una lista
    - **overflow block**, contenenti i record successivi

28

Organizzazione di record in blocchi

- Un file può essere visto come una collezione di record
- Poiché i dati sono trasferiti in blocchi tra la MS e la MM, è però importante assegnare i record ai blocchi in modo tale che uno stesso blocco contenga record tra loro interrelati
- Se si riesce a memorizzare sullo stesso blocco record che sono spesso richiesti insieme si risparmiano accessi a disco
  - es.: nell'organizzazione fixed-length per i record a lunghezza variabile, può essere utile memorizzare tutti i record, relativi allo stesso record a lunghezza variabile, nello stesso blocco
- Se si eseguono molte modifiche, un blocco finisce per contenere record di liste diverse

29

Organizzazione di record in blocchi

- Non è sempre possibile organizzare i record in blocchi in modo tale che un blocco sia completamente occupato da record
  - in ogni blocco si ha una certa quantità di spazio inutilizzato
- si può memorizzare parte di un record in un blocco e la parte rimanente in un altro blocco
- **organizzazione spanned**: i record possono essere memorizzati su più di un blocco
  - inevitabile se la dimensione del record supera quella del blocco
- **organizzazione unspanned**: ogni record è memorizzato su un unico blocco
  - preferibile nel caso di file con record a lunghezza fissa

30

Organizzazione di record in blocchi

- **Tecniche per l'allocazione dei blocchi di un file su disco:**
  - **allocazione contigua**: i blocchi del file sono allocati in blocchi di disco contigui
    - lettura dell'intero file molto efficiente
    - espansione del file difficile
  - **allocazione linkata**: ogni blocco di un file contiene un puntatore al successivo blocco del file
    - espansione del file facile
    - lettura dell'intero file lenta
  - **utilizzo di bucket** (cioè un insieme di blocchi) per gruppi di record tra loro collegati (ad esempio, tutti gli impiegati di un certo dipartimento)

31

Organizzazione di record in blocchi

- É possibile avere bucket che occupano più blocchi: i blocchi di uno stesso bucket sono collegati tra loro (block header memorizza il puntatore al prossimo blocco)
- se un bucket aumenta di dimensione si allocano nuovi blocchi
  - i blocchi liberi sono collegati per poterli riusare in caso di nuove inserzioni nello stesso bucket
- é meglio non riusare i blocchi liberi di un bucket per memorizzare record di un altro bucket
  - i blocchi di uno stesso bucket sono memorizzati nello stesso cilindro

32

Gestione del buffer

- L'obiettivo principale delle strategie di memorizzazione è di minimizzare gli accessi a disco
- Un altro modo è di mantenere più blocchi possibile in MM
- Si usa un buffer che permette di tenere in MM copia di alcune pagine di disco
- Il buffer è organizzato in pagine, che hanno la stessa dimensione dei blocchi

33

Gestione del buffer

- Quando una pagina della MS è presente nel buffer, il DBMS può effettuare le sue operazioni di lettura e scrittura direttamente su di essa
- I tempi di accesso alla MS sono dell'ordine di millesimi di secondo mentre quelli di accesso alla MM sono dell'ordine di milionesimi di secondo
- Accedere alle pagine nel buffer invece che alle corrispondenti pagine su disco influenza notevolmente le prestazioni

34

Gestione del buffer

- Il buffer manager di un DBMS usa alcune politiche di gestione che sono più sofisticate delle politiche usate nei SO:
  - le politiche di LRU non sempre sono le più adatte per i DBMS
  - per motivi legati alla gestione del recovery in alcuni casi un blocco non può essere trasferito su disco (in tal caso il blocco è detto **pinned**)
  - per motivi legati alla gestione del recovery in alcuni casi è necessario forzare un blocco su disco
  - un DBMS è in grado di predire meglio di un SO il tipo dei futuri riferimenti

35

Gestione del buffer: Esempio

- Operazione di join Impiegati |x| Dipartimenti (assumendo che le due relazioni siano in due file diversi)
- relazione Impiegati
  - una volta che una tupla della relazione è stata usata non è più necessaria
  - non appena tutte le tuple di un blocco sono state esaminate il blocco non serve più
    - strategia *toss-immediate*

36

Gestione del buffer: Esempio

- relazione Dipartimenti
  - il blocco più recentemente acceduto sarà riferito di nuovo solo dopo che tutti gli altri blocchi saranno stati esaminati
  - la strategia migliore per il file Dipartimenti è di rimuovere l'ultimo blocco esaminato
    - strategia most recently used - MRU
  - è però necessario eseguire il pin del blocco correntemente esaminato fino a che si siano esaminate tutte le tuple; quindi si può rendere il blocco unpinned

37

Mapping di relazioni a file

- Per DBMS di piccole dimensioni (es. per PC) una soluzione spesso adottata è di memorizzare ogni relazione in un file separato
- nel caso di DBMS large-scale questa strategia di base deve essere estesa (molto spesso il DBMS deve poter allocare in modo opportuno i record ai blocchi per minimizzare le operazioni di I/O)
  - una strategia frequente è di allocare per il DBMS un unico grosso file, in cui sono memorizzate tutte le relazioni
  - la gestione di questo file è lasciata al DBMS

38

Clustering

- Si consideri la seguente interrogazione:

```
SELECT Imp#, Nome, Sede
FROM Impiegati, Dipartimenti
WHERE Impiegati.Dip# = Dipartimenti.Dip#
```

- una strategia di memorizzazione efficiente è basata sul raggruppamento (clustering) delle tuple che hanno lo stesso valore dell'attributo di join
- il clustering può rendere inefficiente l'esecuzione di altre interrogazioni
  - es. SELECT * FROM Dipartimenti

39

Clustering

10	Edilizia Civile	1100	D1	7977		
7782	Neri	ingegnere	01-Giu-81	2,450.00	200.00	10
7839	Dare	ingegnere	17-Nov-81	2,600.00	300.00	10
7934	Milli	ingegnere	23-Gen-82	1,300.00	150.00	10
7977	Verdi	dirigente	10-Dic-80	3,000.00	?	10
20	Ricerche	2200	D1	7566		
7369	Rossi	ingegnere	17-Dic-80	1,600.00	500.00	20
7566	Rosi	dirigente	02-Apr-81	2,975.00	?	20
7788	Scotti	segretaria	09-Nov-81	800.00	?	20
7876	Adami	ingegnere	23-Set-81	1,100.00	500.00	20
7902	Fordi	segretaria	03-Dic-81	1,000.00	?	20
30	Edilizia Stradale	5100	D2	7698		
7499	Andrei	tecnico	20-Feb-81	800.00	?	30
7521	Bianchi	tecnico	20-Feb-81	800.00	100.00	30
7654	Martini	segretaria	28-Set-81	800.00	?	30
7698	Blacchi	dirigente	01-Mag-81	2,850.00	?	30
7844	Turni	tecnico	08-Set-81	1,500.00	?	30
7900	Gianni	ingegnere	03-Dic-81	1,950.00	?	30

40

Strutture ausiliarie di accesso

- Spesso le interrogazioni accedono solo un piccolo sottoinsieme dei dati
- Per risolvere efficientemente le interrogazioni può essere utile allocare delle strutture ausiliarie che permettano di determinare direttamente i record che verificano una data query (senza scandire tutti i dati)
- I meccanismi più comunemente usati dai DBMS sono: indici, funzioni hash

41

Strutture ausiliarie di accesso

- Ogni tecnica deve essere valutata in base a:
  - tempo di accesso
  - tempo di inserzione
  - tempo di cancellazione
  - occupazione di spazio
- Molto spesso è preferibile aumentare l'occupazione di spazio se questo contribuisce a migliorare le prestazioni
- Si usa il termine chiave di ricerca per indicare un attributo o insieme di attributi usati per la ricerca (diverso dalla chiave primaria)

42

Strutture ausiliarie di accesso

- Una ricerca può essere effettuata per:
  - chiave primaria: il valore della chiave identifica un unico record
    - Es. il contribuente con codice fiscale GRRGNN69R48
  - chiave secondaria: il valore della chiave può identificare più record
    - Es. i contribuenti di Genova
  - intervallo di valori (sia per chiave primaria che per secondaria)
    - Es. i contribuenti con reddito compreso tra 60 e 90 milioni
  - combinazioni delle precedenti
    - Es. i contribuenti di Genova e La Spezia con reddito compreso tra 60 e 90 milioni

43

Strutture ausiliarie di accesso

- Per effettuare la ricerca in modo più efficiente si può pensare di mantenere il file ordinato secondo il valore di una chiave di ricerca
- in questo caso però la ricerca su altri campi è inefficiente

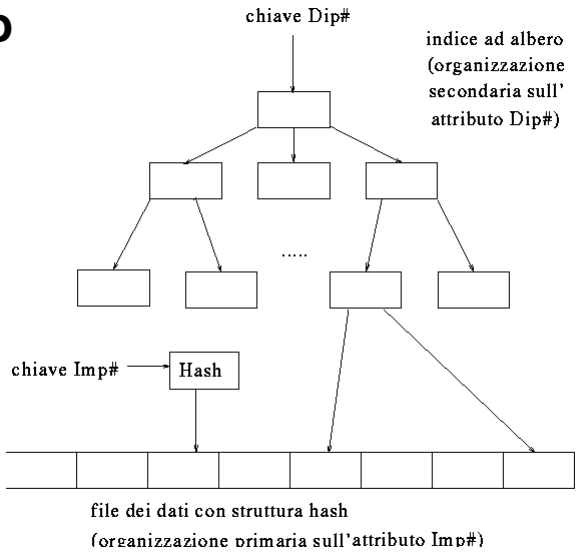
44

Strutture ausiliarie di accesso

- **Organizzazioni primarie**:
  - impongono un criterio di allocazione dei dati (organizzazioni ad albero o hash)
- **organizzazioni secondarie**:
  - uso di **indici** (separati dal file dei dati) normalmente organizzati ad albero
- in generale si hanno a disposizione più modalità (cammini) di accesso ai dati

45

Strutture ausiliarie di accesso



46

Indici

- **Idea base**: associare al file dei dati una “tabella'' nella quale l'entrata i-esima memorizza una coppia (k_i ,r_i) dove:
  - k_i è un valore di chiave del campo su cui l'indice è costruito
  - r_i è un riferimento al record (eventualmente il solo) con valore di chiave k_i
    - il riferimento può essere un indirizzo (logico o fisico) di record o di blocco

47

Indici: esempio

- File dei dati:

chiavi	c5	c2	c11	c7	c4
indirizzi	0	8	16	32	48

- indice:

chiave k_i	indirizzo r_i
c2	8
c4	48
c5	0
c7	32
c11	16

48

Indici

- Le diverse tecniche differiscono nel modo in cui organizzano l'insieme di coppie (k_i ,r_i)
- vantaggio nell'uso di un indice:
  - la chiave è solo parte dell'informazione contenuta in un record, quindi l'indice occupa meno spazio del file dei dati
- un indice può comunque raggiungere notevoli dimensioni che determinano problemi di gestione simili a quelli del file dei dati

49

Indici

- Esempio:
  - Indice per un file di 50k record, in cui i valori di chiave sono stringhe di 20 byte e i puntatori sono di 4 byte, richiede circa 1,2Mb
    - ogni entrata nell'indice: 20 + 4 byte
    - numero max entrate: 50 k
    - totale: 24 * 50 K = 1,2 Mb

50

File sequenziali ad indice

- per applicazioni che richiedono sia accessi sequenziali che accessi casuali ai singoli record
  - file sequenziale ad indice = file sequenziale + file indice
- **file sequenziale**: elaborazione efficiente di record ordinati su una chiave di ricerca
  - record collegati tramite puntatori in base all'ordinamento
  - record memorizzati (possibilmente) in base all'ordinamento (si minimizza il numero di blocchi acceduti)
  - dopo molte modifiche, l'ordinamento dato dalla chiave di ricerca non coincide con l'ordinamento fisico
    - riorganizzazione del file
- strutture ad indice per rendere più efficiente l'accesso casuale

51

Tipi di indice

- Gli indici possono essere classificati rispetto a diverse dimensioni:
  - unicità valori chiave di ricerca
  - ordinamento dei record nel file di dati
  - numero di entrate nell'indice
  - numero di livelli

52

Unicità dei valori di chiave

- **Indice su chiave primaria**:
  - indice su un attributo che è chiave primaria per la relazione (a ogni entrata dell'indice corrisponde un solo record)
- **Indice su chiave secondaria**:
  - indice su un attributo che non è chiave primaria per la relazione (ad ogni entrata dell'indice possono corrispondere più record)

53

Ordinamento dei record nel file dei dati

- **Indice clusterizzato** (o indice primario):
  - indice sull'attributo secondo i cui valori il file dei dati è mantenuto ordinato
- **indice non clusterizzato** (o indice secondario):
  - indice su un attributo secondo i cui valori il file dei dati non è mantenuto ordinato

54

Numero di coppie nell'indice

- **Indice denso**
  - indice il cui numero di entrate (ki, ri) è pari al numero di valori di ki
- **Indice sparso**
  - indice il cui numero di entrate (ki, ri) è minore del numero di valori di ki

55

Numero di livelli

- **Indice a singolo livello**
  - indice organizzato su un singolo livello
- **Indice multilivello**
  - indice organizzato su più livelli

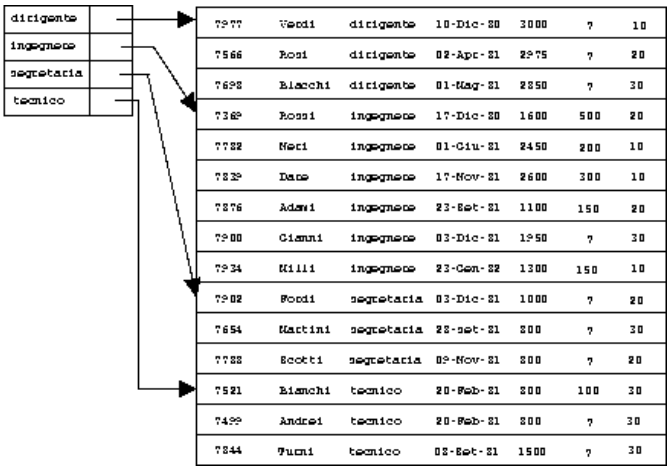
56

Indici densi e sparsi

- **Indice denso**: l'indice contiene un'entrata per ogni valore della chiave di ricerca nel file
  - *Ricerca*: scansione per trovare il record con valore chiave uguale al valore cercato
  - recupero dati fino a che il valore non cambia
- **Indice sparso**: le entrate dell'indice sono create solo per alcuni valori della chiave.
  - *Ricerca*: scansione fino a trovare il record con il più alto valore della chiave che sia minore o uguale al valore cercato
  - scansione sequenziale del file dei dati fino a trovare il record cercato

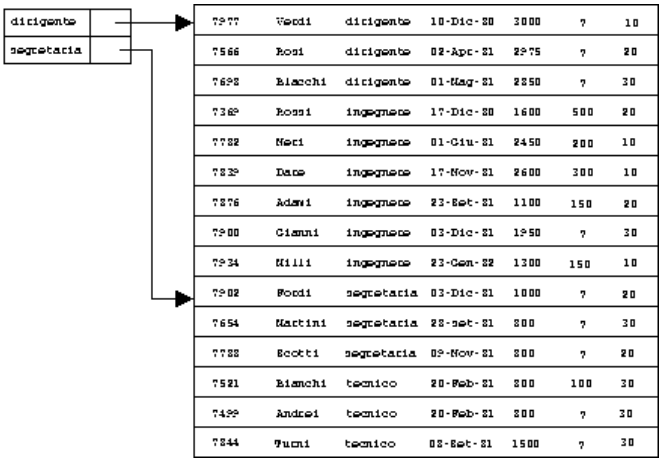
57

Indice denso



58

Indice sparso



59

Indici densi e sparsi

- Un indice denso consente una ricerca più veloce, ma impone maggiori costi di aggiornamento
- Un indice sparso è meno efficiente ma impone minori costi di aggiornamento
- Poiché molto spesso la strategia è di minimizzare il numero di blocchi trasferiti, un compromesso spesso adottato consiste nell'avere una entrata nell'indice per ogni blocco

60

Indici multilivello

- Un indice anche se sparso può essere di dimensioni notevoli
- Esempio:
  - file di 100000 record, con 10 record per blocco, richiede un indice con 10000 entrate
  - se ogni blocco contiene 100 entrate dell'indice: 100 blocchi

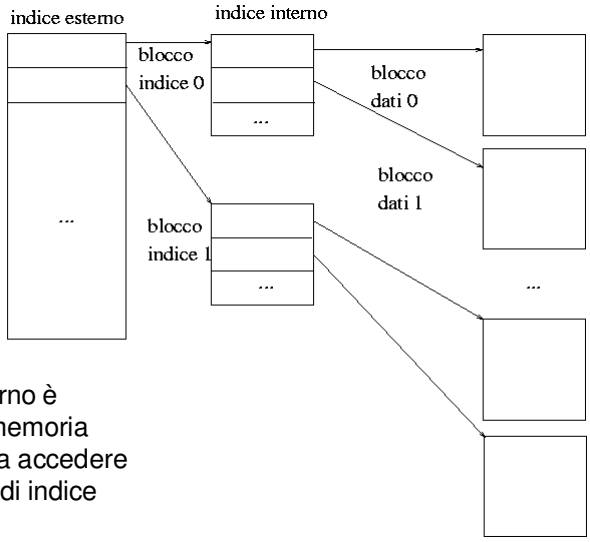
61

Indici multilivello

- Se l'indice è piccolo, può essere tenuto in memoria principale
- molto spesso, però, è necessario tenerlo su disco e la scansione dell'indice può richiedere parecchi trasferimenti di blocchi
  - circa 7 nel nostro esempio, se si utilizza ricerca binaria
- Soluzione:
  - si tratta l'indice come un file e si alloca un indice sparso sull'indice stesso
  - si parla di **indice sparso a due livelli**

62

Indici multilivello



- se l'indice esterno è mantenuto in memoria principale basta accedere un solo blocco di indice

63

Indici clusterizzati

- In un file sequenziale ad indice normalmente si mantiene un solo indice
- se si vogliono mantenere più indici, l'indice la cui chiave è usata per mantenere l'ordinamento dei record nel file è detto **indice primario**, o clusterizzato
- gli altri indici sono detti **indici secondari**, o non clusterizzati

64

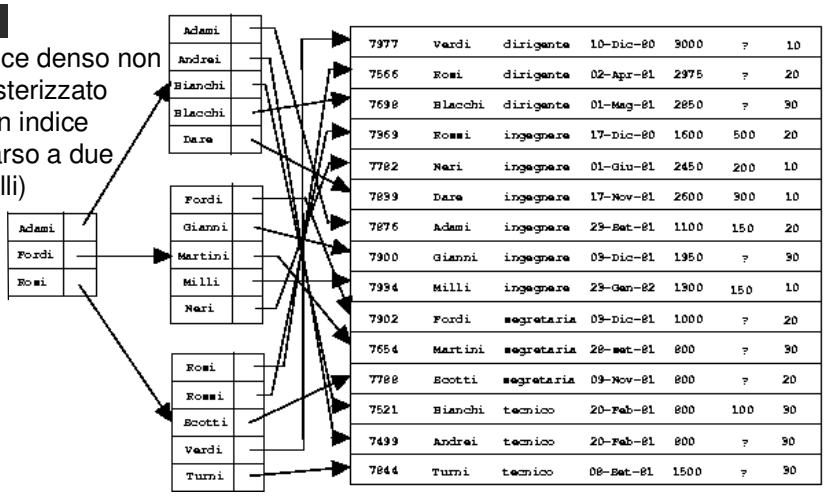
Indici clusterizzati

- É preferibile usare indici densi per gli indici secondari anziché usare indici sparsi
  - il file dei dati non è ordinato in basi alla chiave dell'indice
- l'uso di più indici secondari rende l'esecuzione delle interrogazioni più efficiente, ma rende più costosi gli aggiornamenti

65

Indici clusterizzati: esempio

indice denso non clusterizzato (con indice sparso a due livelli)



66

Tecniche

- Le strutture per MS differiscono da quelle per MM perché si cerca di minimizzare il numero di blocchi acceduti (che determina il costo della ricerca)
  - basate su alberi
    - B-tree e varianti
  - basate su tabelle hash

67

B-Alberi

- i B-alberi sono organizzazioni ad albero bilanciato, utilizzate come strutture di indicizzazione per dati in memoria secondaria
- requisiti fondamentali per indici per memoria secondaria:
  - **bilanciamento**: l'indice deve essere bilanciato rispetto ai blocchi e non ai singoli nodi (è il numero di blocchi acceduti a determinare il costo I/O di una ricerca)
  - **occupazione minima**: è importante che si possa stabilire un limite inferiore all'utilizzazione dei blocchi, per evitare una sotto-utilizzazione della memoria
  - **efficienza di aggiornamento**: le operazioni di aggiornamento devono avere costo limitato

68

B-alberi

- garantiscono un'occupazione di memoria almeno del 50% (almeno metà di ogni pagina allocata è effettivamente occupata)
- consentono di effettuare l'operazione di ricerca con costo, nel caso peggiore, logaritmico nella cardinalità dell'indice (cioè nel numero di valori distinti della chiave di ricerca)
- in un B-albero ogni nodo ha al più *m* figli, dove *m* è l'unico parametro dipendente dalle caratteristiche della memoria, cioè dalla dimensione del blocco

69

B-alberi

- Un B-albero di ordine *m* ($m \geq 3$) è un albero bilanciato che soddisfa le seguenti proprietà:
  - ogni nodo contiene al più m - 1 elementi
  - ogni nodo contiene almeno $\lceil m/2 \rceil - 1$ elementi,
  - la radice può contenere anche un solo elemento
- ogni nodo non foglia contenente j elementi ha j +1 figli
- ogni nodo ha una struttura del tipo:

  $p_0(k_1, r_1)p_1(k_2, r_2)p_2 \ldots p_{j-1}(k_j, r_j)p_j$

  dove j è il numero degli elementi del nodo

70

B-alberi

In ogni nodo

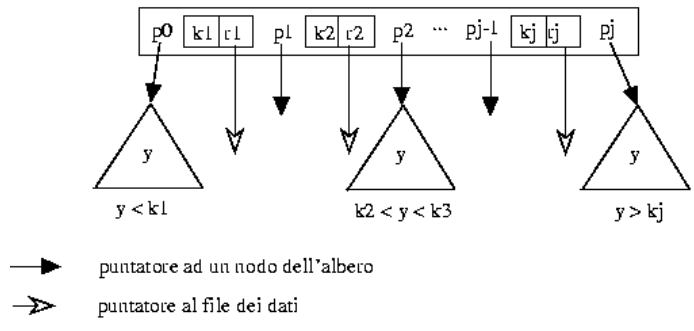
$$p_0(k_1, r_1)p_1\ (k_2, r_2)p_2\ \ldots\ p_{j-1}(k_j, r_j)p_j$$

- $k_1, \ldots, k_j$ sono chiavi ordinate, cioè $k_1 < k_2 < \ldots < k_j$
- nel nodo sono presenti $j + 1$ riferimenti ai nodi figli $p_0, \ldots, p_j$ e j riferimenti al file dei dati $r_1, \ldots, r_j$
- per ogni nodo non foglia, detto $K(p_i)$ $(i = 1, \ldots, j)$ l'insieme delle chiavi memorizzate nel sottoalbero di radice p_i, si ha:
  - $\forall y \in K(p_0), y < k1$
  - $\forall y \in K(p_i), k_i < y < k_{i+1}, i = 1, \ldots, j - 1$
  - $\forall y \in K(p_j), y > k_j$

71

B-alberi

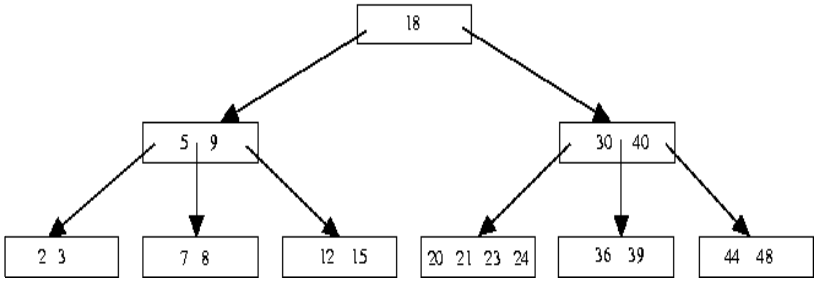
- Formato di un nodo di un B-albero



72

B-alberi

- Esempio di B-albero di ordine 5



73

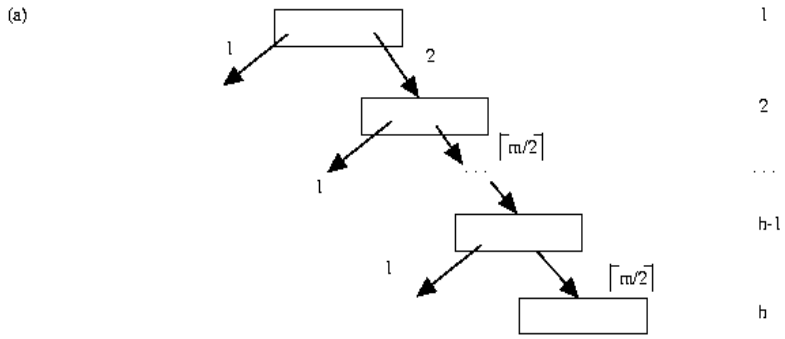
B-alberi

- **altezza** = numero di nodi che compaiono in un cammino dalla radice ad un nodo foglia
- i B-alberi permettono di prevedere con sufficiente approssimazione l'altezza media dell'albero in funzione delle chiavi presenti
  - si possono stimare i costi della ricerca
- $\mathbf{b_{min}}$ = numero minimo di nodi che un B-albero di altezza *h* può contenere
- $\mathbf{b_{max}}$ = numero massimo di nodi che un B-albero di altezza *h* può contenere

74

B-alberi

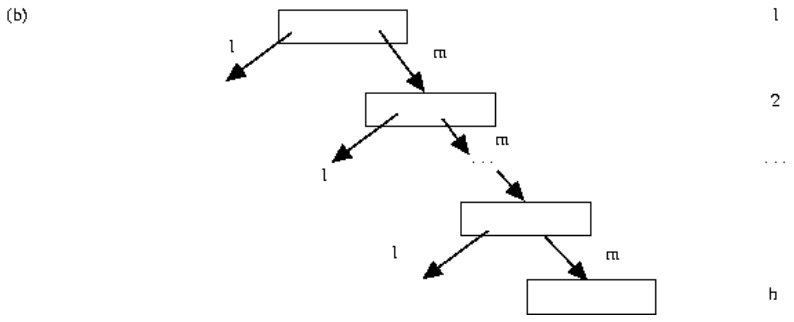
- Numero minimo di nodi in albero di altezza *h*



75

B-alberi

- Numero massimo di nodi in albero di altezza *h*



76

B-alberi

- numero minimo di nodi

  $\mathbf{b_{min}} = 1+2+2\lceil m/2\rceil+2\lceil m/2\rceil^2+\ldots+2\lceil m/2\rceil^{h-2}=$

  $$= 1+2\frac{\lceil m/2\rceil^{h-1}-1}{\lceil m/2\rceil-1}$$

  si ricorda che $\sum_{i=0}^{k-1} x^i = \frac{x^k-1}{x-1}$

77

B-alberi

- numero massimo di nodi

  $\mathbf{b_{max}} = 1+m+m^2+\ldots+m^{h-1} = \frac{m^h-1}{m-1}$

- $\mathbf{N_{min}}$ = numero minimo di chiavi in un albero di altezza *h*
- $\mathbf{N_{max}}$ = numero massimo di chiavi in un albero di altezza *h*

78

B-alberi

- **minimo numero di chiavi**: il numero di nodi è b_{min} e quindi ogni nodo contiene $\lceil m/2 \rceil - 1$ chiavi e la radice contiene una sola chiave

$$N_{\min} = 1 + (\lceil m/2 \rceil - 1)(b_{\min} - 1) =$$

$$= 1 + (\lceil m/2 \rceil - 1) 2 \frac{\lceil m/2 \rceil^{h-1} - 1}{\lceil m/2 \rceil - 1} = 2\lceil m/2 \rceil^{h-1} - 1$$

79

B-alberi

- **massimo numero di chiavi**: il numero dei nodi è b_{max} e quindi ogni nodo, compresa la radice, contiene m - 1 chiavi

$$N_{\max} = (m-1) b_{\max} = (m-1) \frac{m^h - 1}{m-1} = m^h - 1$$

80

B-alberi

- se N indica il numero di chiavi di un B-albero, si ha:

$$2\lceil m/2 \rceil^{h-1} - 1 \leq N \leq m^h - 1$$

$$\log_m (N+1) \leq h \leq 1 + \log_{\lceil m/2 \rceil} \frac{N+1}{2}$$

81

B-alberi

- Altezza di un B-albero in funzione del numero dei nodi e dell'ordine, supponendo che le chiavi siano di 10 byte ed i puntatori di 4 byte

dim pagina	m	$N = 1000$		$N = 10000$		$N = 100000$		$N = 1000000$	
		h_{min}	h_{max}	h_{min}	h_{max}	h_{min}	h_{max}	h_{min}	h_{max}
512	36	1,9	3,2	2,6	3,9	3,2	4,7	3,9	5,5
1024	73	1,6	2,7	2,1	3,4	2,7	4,0	3,2	4,6
2048	146	1,4	2,4	1,8	3,0	2,3	3,5	2,8	4,1
4096	292	1,2	2,2	1,6	2,7	2,0	3,2	2,4	3,6

82

B-alberi: ricerca di un elemento

- una volta trasferita la radice in memoria, si esegue la ricerca tra le chiavi contenute fino a determinare la presenza o l'assenza nel nodo della chiave cercata
  - se la chiave non viene trovata, si continua la ricerca nell'unico sottoalbero del nodo corrente che può contenere l'elemento
  - se invece il nodo è foglia, significa che la chiave non è presente nell'albero
- il costo della ricerca di una chiave è il numero di nodi letti, cioè

$$C_{ricerca}^{\min} = 1$$

$$C_{ricerca}^{\max} = h$$

83

B-alberi: ricerca di un elemento

INPUT: y = il valore della chiave da cercare
root = il puntatore alla radice dell'albero
OUTPUT: trovata = True se la chiave è presente,
trovata= False altrimenti
METODO:
P(p) = il nodo puntato da p
k1, . . . , kj = le chiavi in P(p)
p0, . . . , pj = i puntatori ai figli in P(p)

```
p := root;
trovata := False;
While (p ≠ nil) and (not trovata) Do:
        If y < k1 Then p := p0
        Else If ∃i  y = ki Then trovata := True
                Else If ∃ i ki < y < ki+1 Then p := pi
                        Else p := pj
                        endif
                endif
        endif
endwhile
```

84

B-alberi: inserimento

- idea chiave di inserimento e cancellazione:
  - le modifiche partono sempre dalle foglie e l'albero cresce o si accorcia verso l'alto
- Esempio: inserimento
  - non si creano nuovi figli dalle foglie, ma, se necessario, si crea una nuova foglia allo stesso livello delle altre e si propaga un valore di chiave (separatore) verso l'alto
  - i nodi ai livelli superiori non sono necessariamente pieni e quindi possono "assorbire" le informazioni che si propagano dalle foglie
  - la propagazione degli effetti sino alla radice può provocare l'aumento dell'altezza dell'albero

85

B-alberi: inserimento

- l'inserimento richiede prima di tutto un'operazione di ricerca per verificare se l'elemento è già presente nell'albero
- l'inserimento avviene quindi sempre in una foglia - ci possono essere due casi:
  - se la foglia non è piena, si inserisce la chiave e si riscrive la foglia così aggiornata
  - se la foglia è piena, si attiva un processo di **suddivisione** (splitting) che può propagarsi al livello superiore e, nel caso peggiore, propagarsi fino alla radice

86

B-alberi: inserimento - suddivisione

- P un nodo pieno in cui deve essere inserita una chiave
- sequenza ordinata di m entrate che si verrebbero a creare

  $p_0k_1p_1k_2p_2 \ldots k_gp_gk_{g+1}p_{g+1}k_{g+2}p_{g+2} \ldots k_mp_m$

  con $g = \lceil m/2 \rceil - 1$
- si partizionano le chiavi come segue:
  - nel nodo P gli elementi:

    $p_0k_1p_1k_2p_2 \ldots k_gp_g$
  - in un nuovo nodo P' gli elementi:

    $p_{g+1}k_{g+2}p_{g+2} \ldots k_mp_m$

87

B-alberi: inserimento - suddivisione

- nel nodo *Q* padre di P si inserisce l'entrata

  $k_{g+1}p'$

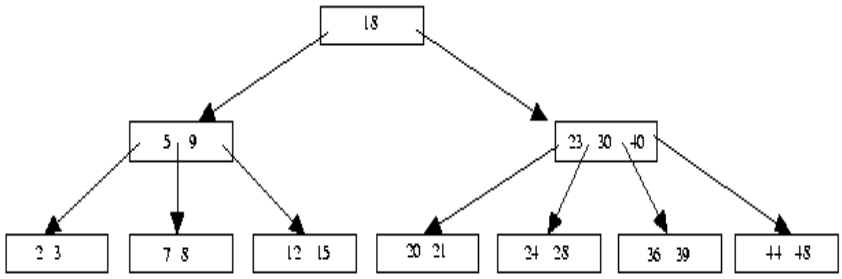
  con p' puntatore al nodo P'
- se anche Q è pieno, allora il processo di splitting deve essere ripetuto
- se si deve sdoppiare la radice, questa diventa

  $pk_{g+1}p'$

  con p puntatore al nodo P (la radice prima dell'inserimento)

88

B-alberi: inserimento - esempio



89

B-alberi: inserimento - costi

- caso migliore (assenza di splitting):
  - si leggono *h* nodi e si riscrive una foglia, quindi

$$C^{\min}_{inserimento} = h + 1$$

- caso peggiore (lo splitting si propaga fino alla radice):
  - si leggono h nodi e se ne riscrivono 2h+1, quindi

$$C^{\max}_{inserimento} = 3h + 1$$

90

B-alberi: cancellazione

- anche in questo caso ci si riconduce ad una cancellazione da un nodo foglia
- viene innanzitutto effettuata la ricerca dell'elemento da cancellare nell'albero
  - se tale elemento non si trova in una foglia, si rimpiazza tale elemento con il valore di chiave più piccolo del sottoalbero puntato dal puntatore alla sua destra

91

B-alberi: cancellazione

- cancellazione da una foglia - si distinguono due casi:
  - se la foglia non è troppo vuota (cioè ha ancora almeno $\lceil m/2 \rceil - 1$ elementi dopo la cancellazione), si cancella la chiave e si riscrive la foglia così aggiornata
  - se la foglia è troppo vuota, si attiva un processo di **concatenazione** o un processo di **bilanciamento**

92

B-alberi: cancellazione - concatenazione

- la concatenazione di due nodi adiacenti P e P' è possibile se i due nodi contengono, complessivamente, meno di m - 1 chiavi
- un nodo con meno di $\lceil m/2 \rceil - 1$ elementi, detto in **underflow**, si combina quindi con un nodo fratello adiacente con al più $\lfloor m/2 \rfloor$ chiavi
- la concatenazione opera in maniera esattamente inversa al processo di suddivisione

93

B-alberi: cancellazione-concatenazione

- situazione iniziale:
  - nodo P in underflow con elementi:

    $p_0 k_1 p_1 k_2 p_2 \ldots k_e p_e$ ($e = \lceil m/2 \rceil - 2$)
  - nodo P' adiacente a destra di P con elementi:

    $p'_0 k_{e+1} p_{e+1} \ldots$
  - nodo Q, padre di P e P', con elementi

    $\ldots k_{t-1} p_{t-1} k_t p_t k_{t+1} p_{t+1} \ldots$

    con p_{t-1} puntatore a P e p_t puntatore a P'

94

B-alberi: cancellazione - concatenazione

- la concatenazione dei due fratelli porta alla seguente situazione:
  - nodo P con elementi:

    $p_0 k_1 p_1 k_2 p_2 \ldots k_e p_e k_t p'_0 k_{e+1} p_{e+1} \ldots$
  - nodo Q con elementi:

    $\ldots k_{t-1} p_{t-1} k_{t+1} p_{t+1} \ldots$

    con p_{t-1} puntatore a P

95

B-alberi: cancellazione - concatenazione

- l'eliminazione della chiave k_t dal nodo padre può innescare a sua volta una concatenazione (o un bilanciamento)
- la concatenazione si può propagare ricorsivamente fino alla radice, causandone l'eliminazione, con conseguente diminuzione dell'altezza dell'albero

96

B-alberi: cancellazione - bilanciamento

- se fra due fratelli adiacenti non si può applicare la concatenazione, allora si distribuiscono tra di essi gli elementi in modo bilanciato
- il bilanciamento interessa anche il nodo padre poichè uno dei suoi elementi viene modificato, ma il numero dei suoi elementi non cambia, quindi il fenomeno non si propaga

97

B-alberi: cancellazione - bilanciamento

- situazione iniziale:
  - nodo P' in underflow con elementi:

    $p'_0 k'_1 p'_1 k'_2 p'_2 \ldots k'_j p'_j$ ($j = \lceil m/2 \rceil - 2$)
  - nodo P adiacente a P' sulla destra con elementi:

    $p_0 k_1 p_1 k_2 p_2 \ldots k_e p_e$
  - nodo Q, padre di P e P', con elementi:

    $\ldots k_{t-1} p_{t-1} k_t p_t k_{t+1} p_{t+1} \ldots$

    con p_{t-1} puntatore a P' e p_t puntatore a P

98

B-alberi: cancellazione - bilanciamento

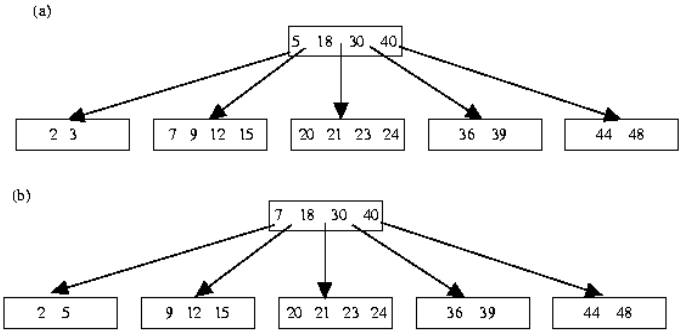
- per bilanciare gli elementi nei due nodi:
  - si considera la lista di chiavi

    $k'_1\ k'_2 \ldots k'_j\ k_t\ k_1\ k_2 \ldots k_e$
  - le prime $\lfloor (e+j)/2 \rfloor$ chiavi rimangono nel nodo P'
  - si sostituisce nel nodo padre la chiave k_t con la chiave in posizione $\lfloor (e+j)/2 \rfloor + 1$
  - le rimanenti $\lceil (e+j)/2 \rceil$ chiavi si mettono in P

99

B-alberi: cancellazione

- (a) Concatenazione [cancellazione di 8]
- (b) Bilanciamento [cancellazione di 3]



100

B-alberi: cancellazione - costi

- **caso migliore** (la cancellazione avviene in una foglia e non si rendono necessari né concatenazione né bilanciamento):
  - si leggono h nodi e si riscrive una foglia, quindi

$$C^{\min}_{cancellazione} = h + 1$$

- **caso peggiore** (tutte le pagine nel percorso di ricerca devono essere concatenate ad eccezione delle prime due, il figlio della radice coinvolto nel percorso viene modificato e la radice viene quindi modificata):
  - si leggono 2h - 1 nodi e se ne riscrivono h+1, quindi

$$C^{\max}_{cancellazione} = 3h$$

101

B-alberi: cancellazione - costi caso peggiore in dettaglio

- R: C(ricerca) + C(esclusa radice) + C(radice)
- W: C(conc) + C(bilanc) + C(radice)
- hp1: ogni lettura un accesso a disco
  - R: h + 3(h-1) + 1 = 4h -2
  - W: (h - 2) + 2 + 1 = h+1
- hp2: se suppongo di concatenare sempre a destra o a sinistra:
  - R: h + 2(h-1) + 1 = 3h - 1
  - W: (h-2) + 2 + 1 = h+1
- hp3: i nodi letti rimangono in memoria + hp2
  - R: h + (h-1) + 0 = 2h -1
  - W: (h-2) + 2 + 1 = h+1

102

B-alberi: bilanciamento nell'inserimento dati

- La struttura di un B-albero dipende dall'ordine in cui vengono inseriti (caricati) i dati
- se le chiavi fossero inserite in maniera ordinata si avrebbe un B-albero con i nodi foglia riempiti tutti a metà tranne al più l'ultimo
- per migliorare l'utilizzo di memoria:
  - B-albero che gestisce l'overflow
  - si usa bilanciamento durante l'inserimento: invece di suddivisione del nodo pieno, bilanciamento con un fratello adiacente fino al suo completo riempimento
  - genera alberi con nodi più pieni, ma comporta maggiori costi di inserimento

103

B+-alberi

- Un B-albero è molto efficiente per la ricerca e la modifica dei singoli record
  - ad esempio, con m = 100 e N = 1000000 la ricerca di una chiave comporta al massimo 4 accessi a disco
- un B-albero non è però particolarmente adatto per elaborazioni di tipo sequenziale nell'ordine dei valori di chiave, né per la ricerca dei valori di chiave compresi in un certo intervallo
  - la ricerca del successore di un valore di chiave può comportare la scansione di molti nodi
- per ovviare a questo problema è stata proposta una variante dei B-alberi, nota come B+-alberi

104

B+-alberi

- **Idea principale**: in un B-albero, i valori di chiave svolgono una duplice funzione:
  - **separatori**: per determinare il cammino da seguire nella ricerca
  - **valori di chiave**: permettono di accedere all'informazione ad essi associata
- nei B+-alberi queste funzioni sono mantenute separate:
  - le foglie contengono tutti i valori di chiave
  - i nodi interni (organizzati a B-albero) memorizzano dei separatori la cui sola funzione è determinare il giusto cammino nella ricerca di una chiave

105

B+-alberi

- i nodi foglia sono inoltre collegati a lista, per facilitare ricerche per intervalli di chiavi o sequenziali (+ puntatore alla testa di tale lista, per accedere velocemente al valore di chiave minimo)
- parziale duplicazione delle chiavi
  - le entrate dell'indice (chiavi e riferimenti ai dati) sono solo nelle foglie
  - la ricerca di una chiave deve individuare una foglia

106

B+-alberi

- il sottoalbero sinistro di un separatore contiene valori di chiave minori del separatore, quello destro valori di chiave maggiori od uguali al separatore
- nel caso di chiavi alfanumeriche facendo uso di separatori di lunghezza ridotta si risparmia spazio

107

B+-alberi

- Un B+-albero di ordine m (m ≥ 3) è un albero bilanciato che soddisfa le seguenti proprietà:
  - ogni nodo contiene al più m- 1 elementi
  - ogni nodo, tranne la radice, contiene almeno $\lceil m/2 \rceil - 1$ elementi, la radice può contenere anche un solo elemento
  - ogni nodo non foglia contenente j elementi ha j +1 figli

108

B+-alberi

- ogni nodo foglia ha una struttura del tipo:

  $(k_1, r_1)(k_2, r_2) \ldots (k_j, r_j)$

  dove:
  - j è il numero degli elementi del nodo
  - $k_1, \ldots, k_j$ sono chiavi ordinate, cioè $k_1 < k_2 < \ldots < k_j$
  - $r_1, \ldots, r_j$ sono j riferimenti al file dei dati
- ogni nodo foglia ha un puntatore al nodo foglia precedente e successivo

109

B+-alberi

- ogni nodo non foglia ha una struttura del tipo:

  $p_0 k_1 p_1 k_2 p_2 \ldots p_{j-1} k_j p_j$

  dove:
  - j è il numero degli elementi del nodo
  - $k_1, \ldots, k_j$ sono chiavi ordinate, cioè $k_1 < k_2 < \ldots < k_j$
  - $p_0, \ldots, p_j$ sono j+1 riferimenti ai nodi figli

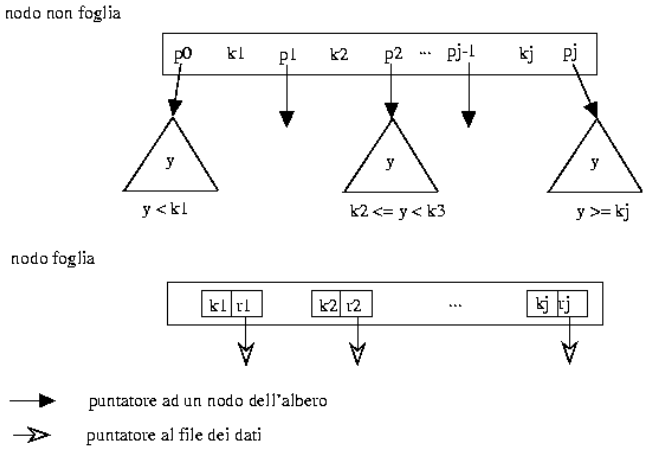
110

B+-alberi

- per ogni nodo non foglia, detto $K(p_i)$ (i = 0, ... , j) l'insieme delle chiavi memorizzate nel sottoalbero di radice p_i, si ha:
  - $\forall y \in K(p_0)$, $y < k1$
  - $\forall\ y \in K(p_i)$, $k_i \leq y < k_{i+1}$, i = 1, . . . , j - 1
  - y 2 $K(p_j)$, $y \geq k_j$
  - ogni k_i, per i = 1, . . . , j è l'elemento minimo di $K(p_i)$

111

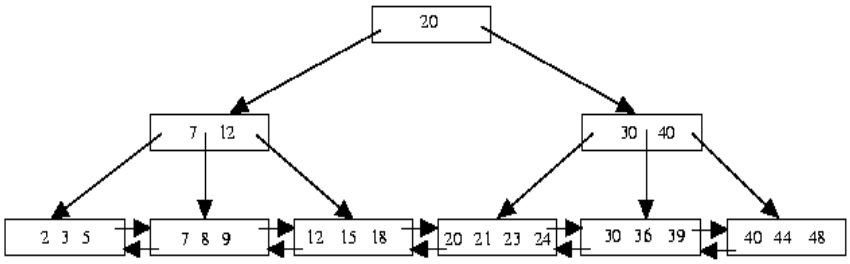
B+-alberi



112

B+-alberi

- Esempio di B+-albero di ordine 5



113

B+-alberi: confronto con B-alberi

- HP: parità di dimensione dei nodi:
- la ricerca di una singola chiave è più costosa in media in un B+-albero (si deve necessariamente raggiungere sempre la foglia per ottenere il puntatore ai dati)
- per operazioni che richiedono il reperimento dei record ordinati in base al valore della chiave o per intervalli di chiave i B+- alberi sono da preferirsi (il collegamento a lista delle foglie elimina la necessità di accedere ai nodi ad altri livelli)
- il B-albero è più conveniente per quanto riguarda l'occupazione di memoria (le chiavi sono memorizzate una volta sola)

114

B- e B+-alberi: terminologia

- **ordine di un B-albero**
  - per noi: numero massimo di figli che un nodo può avere
  - variante: numero minimo di chiavi che un nodo può contenere (formulazione della proposta originaria)
- in alcuni testi, quelli che sono stati indicati come B+-alberi sono chiamati B-alberi
- B+-alberi sono invece detti gli alberi ottenuti dai B-alberi eliminando il puntatore p_0 dai nodi non terminali
- altri testi indicano invece come B^*-alberi una variante dei B-alberi in cui l'utilizzazione dei nodi è almeno pari al 66% invece che al 50%

115

Organizzazioni hash

- Le organizzazioni hash sono usate principalmente per l'organizzazione primaria dei dati
- l'uso di indici ha lo svantaggio che è necessario eseguire la scansione di una struttura dati per localizzare i dati
- questo perché l'associazione

  (chiave, indirizzo)

  è mantenuta in forma esplicita
- un'organizzazione hash utilizza una funzione hash H che trasforma ogni valore di chiave in un indirizzo
- per effettuare la ricerca, data una chiave k, si calcola semplicemente H(k)

116

Organizzazioni hash

- Ogni indirizzo generato dalla funzione hash individua una pagina logica, o bucket
- il numero di elementi che possono essere allocati nello stesso bucket determina la capacità *c* dei bucket
- se una chiave viene assegnata a un bucket che già contiene c chiavi si ha un trabocco (**overflow**)
- la presenza di overflow può richiedere l'uso di un'area di memoria separata, detta **area di overflow**
- l'area di memoria costituita dai bucket indirizzabili dalla funzione hash è detta **area primaria**

117

Organizzazioni hash

- Una funzione hash è detta **perfetta** se per un certo numero di chiavi non produce trabocchi
- una funzione perfetta può sempre essere definita disponendo di un'area primaria con capacità complessiva pari al numero dei record da memorizzare
- dato un alfabeto finito contenente V simboli, V^L è la cardinalità dell'insieme delle chiavi di lunghezza L definibili su tale alfabeto
- se N sono i record da memorizzare, N/V^L viene detta **densità delle chiavi attive**
- in generale, la densità delle chiavi attive è bassa

118

Organizzazioni hash

- Una funzione hash genera M indirizzi (0, . . . ,M - 1) tanti quanti sono i bucket dell'area primaria
- **organizzazione statica**: il valore di M è costante (il dimensionamento dell'area primaria è parte integrante del progetto dell'organizzazione)
- **organizzazione dinamica**: l'area primaria si espande e contrae, per adattarsi al volume effettivo dei dati (si usano più funzioni hash)
- **fattore di caricamento**: $d = \frac{N}{Mc}$ (rapporto tra il numero delle chiavi attive e il massimo numero di record memorizzabili)

119

Organizzazioni hash

- Il progetto di un'organizzazione basata su funzioni hash richiede di specificare:
  - la funzione per la trasformazione della chiave
  - il metodo per la gestione dei trabocchi
  - il fattore di caricamento
  - la capacità delle pagine

120

Organizzazioni hash: trasformazioni della chiave

- Una funzione di trasformazione è un'applicazione surgettiva H dall'insieme delle possibili chiavi all'insieme 0, . . . ,M - 1 dei possibili indirizzi che verifichi le seguenti proprietà:
  - **distribuzione uniforme** delle chiavi nello spazio degli indirizzi (ogni indirizzo deve essere generato con la stessa probabilità)
  - **distribuzione casuale** delle chiavi (eventuali correlazioni tra i valori delle chiavi non devono tradursi in correlazioni tra gli indirizzi generati)
- tali proprietà dipendono dall'insieme delle chiavi su cui si opera e quindi non esiste una funzione universale ottima

121

Organizzazioni hash: trasformazioni della chiave

- Le prestazioni di una funzione hash variano al variare dello specifico insieme di chiavi su cui la funzione opera
- nel caso medio, tuttavia, considerando insiemi di chiavi arbitrari, si osserva che le diverse funzioni hash si comportano effettivamente in modo diverso

122

Organizzazioni hash: trasformazioni della chiave

- consideriamo funzioni hash operanti su insiemi di chiavi intere
- se i valori delle chiavi sono stringhe alfanumeriche, si può associare in modo univoco ad ogni chiave un numero intero, prima di applicare la trasformazione

123

Organizzazioni hash: trasformazioni della chiave

- **Metodo della divisione**: la chiave numerica viene divisa per un numero P e l'indirizzo è ottenuto considerando il resto:

  $$H(k) = k \bmod P$$

  dove mod indica il resto della divisione intera
- per la scelta di P si hanno le seguenti indicazione pratiche:
  - P è il più grande numero primo minore o uguale a M, oppure P è non primo, minore o uguale a M, con nessun fattore primo minore di 20
  - se P < M si deve porre M := P, per non perdere la surgettività
- test sperimentali eseguiti con file con caratteristiche molto diversificate mostrano che, in generale, il metodo della divisione è il più adattabile

124

Organizzazioni hash: trasformazioni della chiave

- Altre possibili funzioni hash sono:
  - **mid square:** la chiave è moltiplicata per se stessa, viene estratto un numero di cifre centrali pari a quello di M-1, e il numero ottenuto è normalizzato a M
  - **shifting:** la chiave è suddivisa in un certo numero di parti, ognuna costituita da un numero di cifre pari a quello di M - 1; tali parti vengono sommate e il numero ottenuto è normalizzato a M

125

Organizzazioni hash: gestione dei trabocchi

- **Scopo**: ridurre al minimo il numero di accessi a bucket necessari a reperire il record cercato
- i metodi possono essere classificati in:
  - **metodi di concatenamento:** basati sull'utilizzo di puntatori; i record dei trabocchi possono essere memorizzati in un'area di memoria separata, detta area di overflow, o nella stessa area primaria
  - **metodi ad indirizzamento aperto:** basati sull'utilizzo di una legge di scansione per determinare altri bucket in area primaria dove memorizzare i record dei trabocchi

126

Organizzazioni hash: gestione dei trabocchi

- **Metodi di concatenamento in area primaria**
  - **approccio a catene confluenti** (coalesced chaining):
    - un trabocco da una pagina primaria i viene memorizzato nella prima pagina non piena i+h e si attiva un riferimento da i a i+h
    - i record per cui H(k) = i o H(k) = i+h vengono memorizzati nella pagina i+h finché questa non diviene satura
    - quando la pagina i+h dà luogo ad un trabocco si procede in modo analogo
    - la lista collega pagine che contengono trabocchi sia da i che da i+h

127

Organizzazioni hash: gestione dei trabocchi

- **Metodi di concatenamento in area primaria**
  - **approccio a catene separate** (separate chaining):
    - vengono collegati a lista i record che collidono (cioè a cui viene assegnato lo stesso indirizzo)
    - mentre nel caso delle catene confluenti le liste collegano le pagine, in questo caso collegano i record
    - quando la funzione di trasformazione associa un record ad una pagina satura, ma occupata da trabocchi, uno di essi si memorizza in un'altra pagina
    - questo metodo migliora la prestazioni, ma complica la gestione dei trabocchi

128

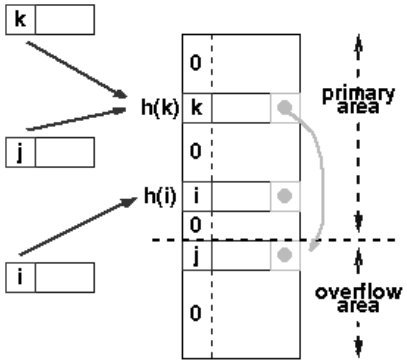
Organizzazioni hash: gestione dei trabocchi

- **Metodi di concatenamento in area separata**
  - i trabocchi vengono memorizzati in un'area di memoria distinta da quella primaria
  - l'area separata è in genere impaginata
  - ogni pagina può essere dedicata a trabocchi provenienti dalla stessa pagina dell'area primaria (si collegano a lista le pagine con trabocchi provenienti dalla stessa pagina)
  - a trabocchi provenienti da pagine diverse (si collegano a lista i trabocchi)
  - la capacità delle pagine dell'area separata non è necessariamente uguale a quella delle pagine dell'area primaria

129

Organizzazioni hash: gestione dei trabocchi

- **Metodi di concatenamento in area separata**



130

Organizzazioni hash: gestione dei trabocchi

- **Metodi di indirizzamento aperto**
  - usano una legge di scansione per visitare l'area primaria, a partire dalla pagina di indirizzo H(k), nella ricerca di una pagina non satura
  - metodo più semplice: scansione lineare (linear probing)

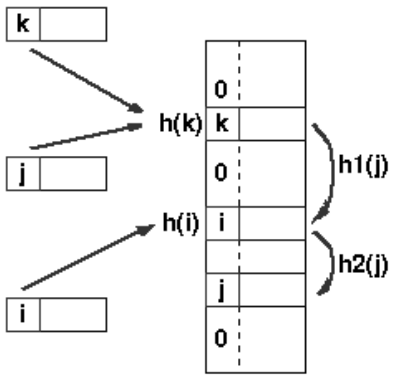
$$H_i(k) = (H_0(k)+si) \bmod M$$

    - si incrementa l'indirizzo iniziale H(k) di una quantità costante s, detta passo
    - se il valore di s non ha divisori in comune con M, i primi M valori di $H_i(k)$ sono tutti i possibili indirizzi delle pagine dell'area primaria

131

Organizzazioni hash: gestione dei trabocchi

- **Metodi di indirizzamento aperto**



132

Organizzazioni hash: gestione dei trabocchi

- **Metodi di indirizzamento aperto**
  - **problema**: agglomerazione primaria
  - invece di distribuirsi uniformemente su tutta l'area primaria i record tendono ad addensarsi in alcune pagine
  - se $H_0(k_1)$ coincide con $H_i(k_2)$, le probabilità di trabocco di tale pagina aumentano e successive applicazioni della legge di scansione restituiranno per entrambe le chiavi le stesse pagine

133

Organizzazioni hash: gestione dei trabocchi

- **Metodi di indirizzamento aperto**
  - esempio: funzione $H_i(k) = (H_0(k)+3i) \bmod 31$
  - le chiavi 1234 e 245 generano rispettivamente le sequenze
    - (25, 28, 0, 3, 6, 9, 12, . . .)

    e
    - (28, 0, 3, 6, 9, 12, . . .)
  - un trabocco dalla pagina 25 aumenta la probabilità di trabocchi a pagina 28, poi a pagina 0, e così via

134

Organizzazioni hash: gestione dei trabocchi

- **Metodi di indirizzamento aperto**
  - per evitare l'agglomerazione primaria è necessario rendere variabile il passo di scansione ricorrendo ad una funzione non lineare come legge scansione
  - **scansione quadratica**
    - $H_i(k) = (H_0(k)+s_1 i+s_2 i^2) \bmod M$
  - **problema**: agglomerazione secondaria, dovuta a chiavi che vengono associate allo stesso indirizzo iniziale

135

Organizzazioni hash: gestione dei trabocchi

- **Metodi di indirizzamento aperto**
  - possibile soluzione: scansione casuale (random probing)
    - $H_1(k) = H_0(k), H_i(k) = (H_{i-1}(k)+s_i) \bmod M$

    dove s_i è una sequenza di numeri casuali diversi nell'intervallo [0,M - 1] generata prendendo la chiave come seme
  - semplice da realizzare, ma disperde troppo i dati sui cilindri (per la natura casuale degli indirizzi generati)
  - si preferisce quindi adottare una legge di scansione lineare con passo unitario, nonostante i fenomeni di agglomerazione

136

Organizzazioni hash: fattore di caricamento

- Data una stima del numero N di record da gestire e fissata la capacità c dei bucket, la scelta di un determinato fattore di caricamento d determina il numero di bucket M in area primaria
- al diminuire di d si riducono i trabocchi ma aumenta lo spazio di memoria occupato dal file
- per ridurre la percentuale di trabocchi, che incide sia sui costi di ricerca che su quelli di aggiornamento, non è consigliabile utilizzare fattori di caricamento elevati

137

Organizzazioni hash: fattore di caricamento

- Valori tipici sono compresi tra 0,75 e 0,85
- per evitare problemi legati alla funzione hash, è necessario scegliere attentamente la dimensione del file M
- tipicamente, quindi, si procede come segue:
  - dato il numero N di record da memorizzare e la capacità c dei bucket, si sceglie un certo fattore di caricamento d
  - si determina M come $M = \frac{N}{dc}$
  - si valuta la percentuale dei trabocchi, se è troppo alta bisogna ridurre d

138

Organizzazioni hash: fattore di caricamento

- la percentuale di trabocchi può essere stimata come segue:
  - con una trasformazione uniforme, la probabilità che un indirizzo si presenti ripetuto x volte è $p^x q^{N-x}$, con $p = \frac{1}{M}$ probabilità che un indirizzo venga generato e $q = 1 - \frac{1}{M}$ probabilità che un indirizzo non venga generato
  - la probabilità che tra gli N indirizzi se ne presenti uno ripetuto x volte è data dalla distribuzione binomiale $P(x) = \binom{N}{X} p^x q^{N-x}$

139

Organizzazioni hash: fattore di caricamento

- Per N grande e p piccolo, quindi per N e M grandi, la distribuzione binomiale è approssimata dalla distribuzione di Poisson

$$P(x) = \frac{m^x e^{-m}}{x!} \text{ con } m = \frac{N}{M} = Np$$

140

Organizzazioni hash: fattore di caricamento

- il numero totale di trabocchi N_t è dato dalla differenza tra il numero totale di registrazioni N e le registrazioni in pagine senza trabocchi ed in pagine con trabocchi, cioè: $N_t = N - (\sum_{x=0}^{c} P(x)Mx + cP_cM)$

  con $P_c = 1 - \sum_{x=0}^{c} P(x)$ probabilità che un indirizzo venga generato più di c volte

141

Organizzazioni hash: fattore di caricamento

- Si ha quindi:

$$\frac{N_t}{N} = 1 - \frac{1}{N}(\sum_{x=0}^{c} P(x)Mx + cP_cM) =$$

$$= 1 - \frac{1}{N}(\sum_{x=0}^{c} P(x)\frac{N}{dc}x + \frac{N}{Md}P_cM) =$$

$$= 1 - (\frac{1}{dc}\sum_{x=0}^{c} P(x)x + \frac{P_c}{d})$$

142

Organizzazioni hash: fattore di caricamento

percentuale di record in overflow per diversi valori di capacità e fattore di caricamento (funzione hash ideale)

c / d	0,5	0,7	0,9	1,0
1	21,31	28,08	34,06	36,79
5	2,48	7,11	13,78	17,55
10	0,44	2,88	8,59	12,51
50	0	0,05	2,04	5,63

143

Organizzazioni hash: capacità delle pagine

- All'aumentare di c, e a parità di fattore di caricamento d, la percentuale di record in overflow diminuisce, sotto le ipotesi di una funzione hash ideale e di gestione degli overflow in area separata
- se ad esempio si devono memorizzare 750 record in 1000 pagine con c = 1 oppure in 500 pagine con c = 2, in entrambi i casi il fattore di caricamento è d = 0,75, ma nel primo caso i trabocchi sono il 29,6%, mentre nel secondo caso i trabocchi sono il 18,7%
- se si mantiene il fattore di caricamento 0,75 e si pone c = 10, le prestazioni migliorano ulteriormente
- per organizzazioni su MS la capacità delle pagine dovrebbe essere superiore a 10

144

Organizzazioni hash: capacità delle pagine

- Poiché la dimensione della pagina fisica (blocco) dipende dal sistema operativo, vengono generalmente utilizzate pagine logiche (bucket)
  - il file dei dati è suddiviso in bucket (uno o più blocchi) e una bucket directory contiene un puntatore ad ogni bucket
- la trasformazione della chiave determina il numero del bucket
- attraverso la bucket directory si determina il primo blocco del bucket che viene poi acceduto per cercare il record
  - se nel primo blocco il record non c'è si passa al successivo e così via

145

Organizzazioni hash dinamiche

- Nelle tecniche hash dinamiche l'allocazione di memoria viene aumentata o diminuita a seconda delle dimensioni del file, senza richiedere riorganizzazioni del file
- gli approcci possono essere suddivisi in due categorie:
  - quelli che fanno uso di strutture ausiliarie (hashing virtuale, hashing estensibile, hashing dinamico)
  - quelli che agiscono esclusivamente sull'area primaria (hashing lineare, hashing spirale)
- in entrambi i casi, la funzione di trasformazione della chiave viene modificata opportunamente quando l'organizzazione si ristruttura

146

Organizzazioni hash dinamiche: hashing virtuale

- **Idea**: raddoppiare l'area primaria quando si verifica un overflow in un bucket, e ridistribuire i record tra il bucket saturo e il bucket a lui corrispondente nell'area raddoppiata, detto suo **buddy**, facendo uso di una nuova funzione hash
- se poi qualche altro bucket nell'area primaria originale diventa saturo e il suo buddy non è ancora in uso, si ridistribuiscono i suoi record tra il bucket stesso e il buddy
- poiché, ad un certo istante, solo alcuni buddy sono effettivamente in uso, è necessario fare uso di una struttura ausiliaria per determinare quale funzione hash utilizzare

147

Organizzazioni hash dinamiche: hashing virtuale

- Inizialmente si alloca per l'area dati un certo numero (anche piccolo, ad esempio 7) M di pagine contigue di capacità c
- si introduce un vettore binario B con tanti elementi quante sono le pagine dell'area dati
- quando una pagina dell'area dati viene utilizzata, il corrispondente elemento di B viene posto a 1
- si utilizza una funzione di trasformazione H_0 che applicata ad una chiave produce un indirizzo compreso tra 0 e M - 1

148

Organizzazioni hash dinamiche: hashing virtuale

- Se inserendo un record con chiave k nella pagina di indirizzo m si genera un trabocco:
  - si raddoppia l'area primaria
  - si raddoppia il vettore B, ponendo a 0 tutte le entrate tra M +1 e 2M - 1
  - si sostituisce H0 con H1 che produce indirizzi compresi tra 0 e 2M - 1
  - si applica H1 a k e a tutte e sole le chiavi dei record nella pagina m; i record verranno ridistribuiti tra la pagina m e la pagina M +m
  - l'entrata di B per la pagina M+m viene messa a 1

149

Organizzazioni hash dinamiche: hashing virtuale

- il metodo richiede l'utilizzo di una serie di funzioni di trasformazione $H_0, H_1, \ldots, H_r$
- H_r restituisce un indirizzo compreso tra 0 e $2^r M - 1$ (l'indice indica il numero di raddoppi subiti dall'area dati)
- le funzioni H devono soddisfare la seguenti proprietà:
  per ogni $j = 0, \ldots, r$:
  $$H_{j+1}(k) = H_j(k) \text{ oppure } H_j(k)+2^j M$$
- una funzione comunemente adottata è
  - $H_r(k) = k \bmod 2^r M$

150

Organizzazioni hash dinamiche: hashing virtuale - ricerca

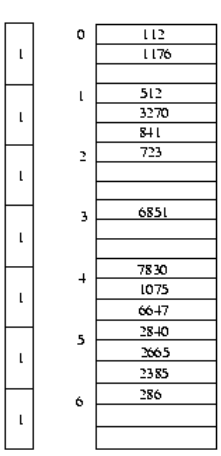
INPUT: k: chiave da cercare
r: numero di raddoppi subiti dall'area dati
OUTPUT: indirizzo corrispondente alla chiave, se la chiave è presente, -1 altrimenti
METODO:

```
function ricerca(r:int; k:int): int;
If r < 0 Then return -1; /* la chiave non `e presente */
Else If B(H_r(k)) = 1 Then return H_r(k)
Else return ricerca(r - 1, k)
endif
endif
```

151

Hashing virtuale: Esempio

Situazione iniziale
M = 7
c = 3



152

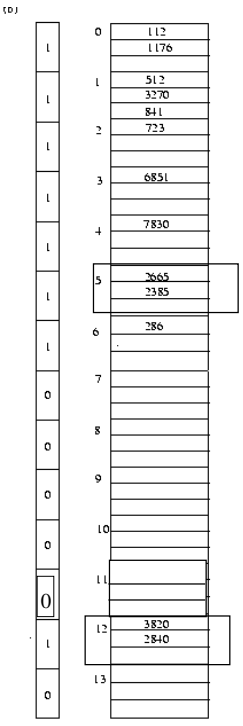
Hashing virtuale: esempio

- si vuole inserire la chiave 3820
  - $H_0(3820) = 5$
  - poiché B(5) = 1, la nuova chiave va memorizzata nella pagina di indirizzo 5
- la pagina è satura per cui occorre raddoppiare l'area dati
- i record della pagina di indirizzo 5 sono divisi tra tale pagina e una nuova pagina di indirizzo 12, mediante la funzione $H_1(k) = k \bmod 14$
- il vettore B viene raddoppiato e i suoi valori aggiornati, da questo momento viene applicata la funzione H_1

153

Hashing virtuale: esempio

Situazione dopo l'inserimento di 3820



154

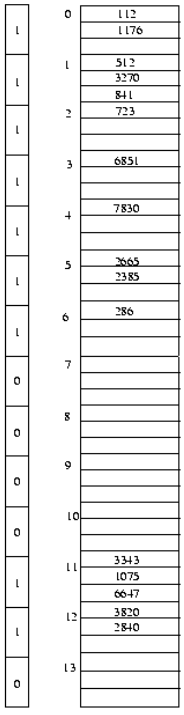
Hashing virtuale: esempio

- si supponga di dover inserire la chiave 3343
  - $H_1(3343) = 11$, ma poiché B(11) = 0, cioé la pagina non è ancora stata utilizzata, si utilizza H_0
  - $H_0(3343) = 4$, la pagina 4 però è satura,
    - si attiva la pagina 11 ponendo ad 1 il bit nel vettore B
    - si trasformano tutte le chiavi della pagina 4, cioè 7830,1075, 6647 più la nuova chiave 3343, mediante la funzione H_1, dividendole così tra le pagine 4 e 11

155

Hashing virtuale: esempio

Situazione dopo l'inserimento di 3343



156

Hashing virtuale: nota

- Si supponga di inserire la chiave k_1 in una pagina $m = H_0(k_1)$ che generi un trabocco
- se applicando la funzione H_1 alla pagina m i record sono nuovamente tutti contenuti nella pagina m (quindi si genera un nuovo trabocco), si raddoppia nuovamente l'area dati e si utilizza la funzione H_2
- si procede in questo modo finché non vengono generati più trabocchi
- sicuramente il procedimento termina
  - sicuramente termina con H_j, con $2^j M$ maggiore di tutti i valori chiave da ridistribuire
- potrebbe però terminare prima la memoria a disposizione, in questo caso il trabocco non può essere gestito

157

Hashing estensibile

- l'hashing estensibile evita di ricorrere a tecniche di raddoppio facendo uso di una struttura ausiliaria, detta **direttorio**
- il direttorio è un insieme di 2p celle, con indirizzi da 0 a 2p-1, con p >= 0 profondità del direttorio
- l'espansione dell'area dati avviene aggiungendo una nuova pagina ogni volta che si tenta di inserire un record in una pagina satura

158

Hashing estensibile

- **idea**: fare uso di una funzione hash che, dato un valore di chiave k_i, restituisce non un indirizzo di bucket, ma una stringa binaria di opportuna lunghezza (ad esempio, 32 bit) detta **pseudochiave**
- la funzione hash associa ad ogni chiave una pseudochiave di cui si considerano i primi p bit per accedere direttamente ad una delle 2p celle, ognuna contenente un puntatore ad un bucket
- ogni bucket ha una profondità locale p'<=p (mantenuta nel bucket) che indica il numero effettivo di bit usati per allocare le chiavi nel bucket stesso

159

Hashing estensibile

- una cella del direttorio contiene il riferimento ad una pagina dell'area dati contenente tutte e sole le registrazioni con pseudochiavi con lo stesso prefisso di lunghezza p'
- per effettuare la ricerca di un record:
  - si estraggono i primi p bit dalla pseudochiave
  - si accede l'entrata dell'indice che corrisponde alla stringa di p bit
  - dall'entrata si determina l'indirizzo della pagina del file che contiene il record cercato

160

Hashing estensibile

- per effettuare l'inserimento di un record:
  - inizialmente si ha un solo bucket e p = p' = 0
  - quando si deve inserire un record in una pagina satura di profondità p', se p = p' innanzitutto si raddoppia il direttorio e si incrementa p di 1, copiando i valori dei puntatori nelle nuove celle corrispondenti

161

Hashing estensibile

  - sia nel caso p = p' che nel caso p' < p, si alloca un nuovo bucket e si distribuiscono le chiavi tra i due bucket (quello saturo e quello appena allocato) facendo uso del p'+1-esimo bit delle pseudochiavi
  - per i due bucket si pone a p'+1 la profondità locale e si aggiorna il direttorio facendo in modo che una sua cella contenga un riferimento al nuovo bucket

162

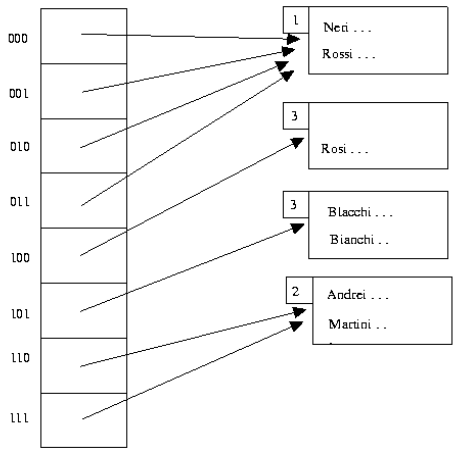
Hashing estensibile: esempio

possibili valori della pseudochiave

Rossi	0010 1101 1111 1011 0010 1100 0011 0000
Andrei	1101 0101 1101 1110 0100 0110 1001 0011
Bianchi	1010 0011 1010 0000 1100 0110 1001 1111
Rosi	1000 0111 1110 1101 1011 1111 0011 1010
Martini	1111 0001 0010 0100 1001 0011 0110 1101
Blacchi	1011 0101 1010 0110 1100 1001 1110 1011
Neri	0101 1000 0011 1111 1001 1100 0000 0001

163

Hashing estensibile: esempio



164

Confronto tra indici e hashing

- se la maggior parte delle interrogazioni ha la forma

  SELECT A1,A2,.....An FROM R WHERE Ai=C

  la tecnica hash è preferibile
- infatti:
  - la scansione di un indice ha un costo proporzionale al logaritmo del numero di valori in R per Ai
  - in una struttura hash il tempo di ricerca è indipendente dalla dimensione della base di dati

165

Confronto tra indici e hashing

- gli alberi sono preferibili se le interrogazioni usano condizioni di range

  SELECT A1,A2,.....An FROM R

  WHERE C1 < Ai < C2
- è infatti difficile determinare funzioni hash che mantengono l'ordine

166

Definizione di cluster e indici in SQL

- la maggior parte dei DBMS relazionali fornisce varie primitive che permettono al progettista della base di dati di definire la configurazione fisica dei dati
- queste primitive sono rese disponibili all'utente come comandi del linguaggio SQL
- non esiste uno standard, ma la sintassi non differisce molto nei vari sistemi per quanto riguarda i comandi principali

167

Definizione di cluster e indici in SQL

- i comandi più importanti sono il comando per la creazione di **indici**, su una o più colonne di una relazione, e il comando per la creazione di **cluster**
- un cluster permette di memorizzare fisicamente contigue le tuple di una o più relazioni che hanno lo stesso valore per una o più colonne, dette colonne del cluster

168

Definizione di cluster e indici in SQL

- il comando per la creazione di un indice in SQL ha il seguente formato:

```
CREATE INDEX Nome Indice
ON Nome Relazione(Lista Nomi Colonne) |
Nome Cluster
[ASC | DESC];
```

  dove
  - Nome Indice è il nome dell'indice che si crea
  - la clausola ON specifica l'oggetto su cui è allocato l'indice

169

Definizione di cluster e indici in SQL

- Tale oggetto può essere:
  - **una relazione**: si devono specificare i nomi delle colonne su cui l'indice è allocato
  - **un cluster**: l'indice viene allocato automaticamente su tutte le colonne del cluster
- l'indice può essere allocato su più colonne
  - i valori della chiave sono ottenuti come concatenazione di tutti i valori di tali colonne, normalmente esiste un limite al numero di colonne (es. in Oracle 16)
- le opzioni ASC e DESC specificano se i valori della chiave dell'indice devono essere ordinati in modo crescente o decrescente
  - ASC è il default

170

Definizione di cluster e indici in SQL - Esempio

- L'indice creato è in genere un B+-tree o una sua variante
- Per allocare un indice sulla colonna stipendio della relazione Impiegati

```
CREATE INDEX idxstipendio
ON Impiegati (stipendio);
```

171

Definizione di cluster e indici in SQL

- Definizione di indici clusterizzati (DB2):

```
CREATE INDEX Nome Indice
ON Nome Relazione(Lista Nomi Colonne)
CLUSTER;
```

- una tabella può avere un solo indice clusterizzato
- se la tabella è non vuota quando si crea l'indice clusterizzato i dati non vengono automaticamente raggruppati (è necessario usare una speciale utility REORG)

172

Definizione di cluster e indici in SQL

- Comando per la creazione di un cluster:

```
CREATE CLUSTER NomeCluster
(NomeCol1 Dominio_1, . . ,NomeColn Dominio_n)
[INDEX | HASH IS   Expr |
    HASHKEYS Intero];
```

- NomeCluster è il nome del cluster che si definisce
- (NomeCol1 Dominio_1, . . ., NomeColn Dominio_n), con n >= 1, è la specifica delle colonne del cluster
  - tale insieme di colonne è detto **chiave del cluster**

173

Definizione di cluster e indici in SQL

- Ogni cluster è sempre associato ad una struttura di accesso ausiliaria
  - **Index**:
    - vengono clusterizzate tuple con lo stesso valore per la chiave del cluster e indicizzate tramite un indice di tipo B+-albero (default)
    - conviene se si hanno frequenti interrogazioni di tipo range sulla chiave del cluster o se le relazioni possono aumentare di dimensione in modo impredicibile
    - *cluster di tipo index*
  - **Hash**:
    - vengono clusterizzate le tuple con lo stesso valore di hash per la chiave del cluster e indicizzate tramite una funzione hash
    - conviene se si hanno frequenti interrogazioni con predicati di uguaglianza su tutte le colonne e se le relazioni sono statiche
    - *cluster di tipo hash*

174

Definizione di cluster e indici in SQL - cluster di tipo hash

- Il DBMS fornisce sempre una funzione hash interna che viene usata come default (spesso basata sul metodo della divisione)
- l'opzione HASHKEYS permette di specificare il numero di valori della funzione hash
- questo valore (se non è un numero primo) viene arrotondato dal sistema al primo numero primo maggiore
- tale valore viene usato come argomento dell'operazione di modulo usata dal sistema per generare i valori della funzione hash

175

Definizione di cluster e indici in SQL - Esempio

- Cluster di tipo indice:

```
CREATE CLUSTER Personale(D# NUMBER);

CREATE INDEX idxpersonnel
ON CLUSTER Personale;
```

- Cluster di tipo hash:

```
CREATE CLUSTER Personale
(D# NUMBER) HASHKEYS 10;
```

dato che l'opzione HASHKEYS ha valore 10, il numero di valori generati dalla funzione hash è 11 (primo numero primo > 10)

176

Definizione di cluster e indici in SQL - cluster di tipo hash

- È possibile cambiare la funzione hash da utilizzare tramite l'opzione HASH IS
- questa opzione può tuttavia essere usata solo se:
  - la chiave del cluster è composta da campi di tipo intero
  - l'espressione deve restituire valori positivi
  - + altre condizioni

177

Definizione di cluster e indici in SQL - cluster di tipo index

- se il cluster è di tipo index, prima di poter eseguire interrogazioni o modifiche è necessario creare un indice sul cluster tramite il comando di CREATE INDEX
- un cluster può includere una o più relazioni
  - **singola relazione**: il cluster è usato per raggruppare le tuple della relazione aventi lo stesso valore per le colonne che sono chiavi del cluster
  - **più relazioni**: il cluster viene usato per raggruppare le tuple di tutte le relazioni aventi lo stesso valore per la chiave del cluster (join efficienti su colonne che sono parte della chiave del cluster)
- una relazione deve essere inserita nel cluster al momento della creazione

178

Definizione di cluster e indici in SQL - Esempio

- per inserire nel cluster Personale le relazioni Impiegati e Dipartimenti

```
CREATE TABLE Impiegati
(Imp# Decimal(4) NOT NULL,
Dip# Decimal(2))
CLUSTER personale (Dip#);

CREATE TABLE Dipartimenti
(Dip# Decimal(4) NOT NULL)
CLUSTER personale (Dip#);
```

179

Definizione di cluster e indici in SQL - Esempio

- i nomi delle colonne delle relazioni su cui si esegue il clustering non devono necessariamente avere lo stesso nome della colonna del cluster, devono però avere lo stesso tipo

180