Esercizio: ricerca binaria sul file ordinati

```
Lower=1;
Upper= B;
While (Upper>=Lower)
        i=(Lower+Upper)/ 2
        lCarico in memoria  il blocco i
        Leggo t0,  il primo record del blocco i
        if (k<t0.ID)
                Upper=i-1
        else
                leggo tn, l'ultimo record del blocco i
                If (k> t.ID)
                        Lower=i+1;
                else
                        il blocco con il record  è in memoria
                        exit
endWhile
```

Quanti accessi in memoria secondaria sono necessari?

Costo ricerca su file ordinato

- Si assuma un file ordinato di B blocchi: quanti accessi in memoria secondaria sono necessari per una ricerca binaria?

Numero accessi: $\lceil \log_2 B \rceil$

Costo ricerca in termini di numero accessi

- Si supponga un file ordinato su campo ID con r=300.000 record memorizzati su disco con blocco B=4096 di lunghezza. I record hanno lunghezza fissa R=100 byte e sono memorizzati in modo unspanned.

Mediamente, quanti accessi sono richiesti per ricerca sul campo di ordinamento?

- Fattore di blocco

$$\lfloor 4096/100 \rfloor = 40$$

- Numero blocchi per memorizzare i record:

$$\lceil 300.000/40 \rceil = 7500$$

- Numero di accessi ai blocchi per ricerca binaria:

$$\lceil \log_2 7500 \rceil = 13$$

Costo ricerca in termini di numero accessi

r=300.000 record memorizzati su disco con blocco B=4096 di lunghezza. I record hanno lunghezza fissa R=100 byte e sono memorizzati in modo unspanned.

- File heap: 3750 accesi
- File ordinato sul campo di ricerca: 13 accessi

Organizzazione dei record all'interno del file: strutture primarie: accesso calcolato

Strutture di ricerca interne al programma: accesso calcolato

- I record sono inseriti in base al valore di un campo chiave (non per forza chiave primaria)
- La locazione dei dati dipende dal **valore** -> *accesso associativo*

ESEMPIO

Si consideri un azienda dove gli unici impiegati hanno matricola da 1 a 100.

- *memorizzo i record impiegati ordinati per il campo matricola*
- *Il **valore della matricola** indica la posizione del record e del blocco*
  - *Es. con fattore blocco 2, il record dell'impiegato con matricola 11 è il primo record del 6° blocco*

Abbiamo un accesso diretto ad un insieme di record sulla base del valore di un campo (**chiave**)

Organizzazione dei record all'interno del file: strutture primarie: accesso calcolato

- Se il numero dei valori della chiave è paragonabile al numero di record effettivi
  - Il valore della chiave rappresenta l'indice di un array.
  - Es. matricola=11, elemento 11-esimo dell'array
- Se il numero dei valori della chiave è molto più ampio dei record effettivi?
  - Matricola impiegati codificata in 6 cifre (1 milione di possibili valori), ma solo 100 impiegati
  - Array sparso!
- Volendo continuare ad usare qualcosa di simile ad un array, ma senza sprecare spazio, possiamo pensare di trasformare i valori della chiave in possibili indici di un array attraverso una:

funzione di hash

Organizzazione dei record all'interno del file: strutture primarie: accesso calcolato (hash)

- **Funzione di hash** (strutture interna)
  - associa ad ogni valore della chiave un "indirizzo" ad uno spazio di memoria di dimensione.
  - poiché il numero di possibili chiavi è molto maggiore del numero di possibili indirizzi esiste la possibilità di collisioni (chiavi diverse che corrispondono allo stesso indirizzo)

Un esempio

- Matricola 6 cifre
- 40 record

M
60600
66301
205751
205802
200902
116202
200604
66005
116455
200205
201159
205610
201260
102360
205460
205912
205762
200464
205617
205667

M
200268
205619
210522
205724
205977
205478
200430
210533
205887
200138
102338
102690
115541
206092
205693
205845
200296
205796
200498
206049

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: matricola Mod 50

M	M mod 50
60600	
66301	
205751	
205802	
200902	
116202	
200604	
66005	
116455	
200205	
201159	
205610	
201260	
102360	
205460	
205912	
205762	
200464	
205617	
205667	

M	M mod 50
200268	
205619	
210522	
205724	
205977	
205478	
200430	
210533	
205887	
200138	
102338	
102690	
115541	
206092	
205693	
205845	
200296	
205796	
200498	
206049	

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: matricola Mod 50

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: matricola Mod 50
  - Collisioni?

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: matricola Mod 50
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: matricola Mod 50
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2

  Quanto costano le collisioni

  in termini di accessi?

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: matricola Mod 50
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2

Quanto costano le collisioni in termini di accessi?

In caso di collisione a n elementi,

l'accesso al primo costa 1,

quello al secondo costa 2,

cosi via fino al costo pari a n per l'n-esimo

- Numero medio di accessi alla struttura dati per un generico elemento?

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

Matricola 6 cifre

40 record

Array di 50 elementi

Funzione hash: matricola Mod 50

1 collisione a 4

2 collisioni a 3

5 collisioni a 2

Quanto costano le collisioni in termini di accessi?

In caso di collisione a n elementi, l'accesso al primo costa 1, quello al secondo costa 2, cosi via fino al costo pari a n per l'n-esimo

- Numero medio di accessi alla struttura dati per un generico elemento?

1,425

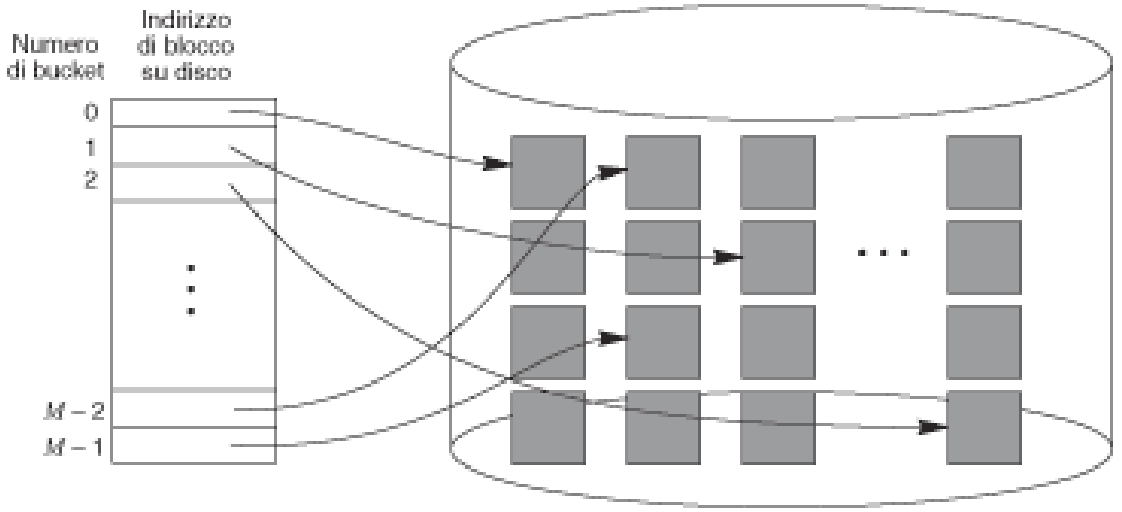
M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Organizzazione dei record all'interno del file: strutture primarie: accesso calcolato (hash)

Funzione hash applicate all'organizzazione di file:

- Il valore hash del campo chiave di un record indica il bucket in cui il record è memorizzato:
  - Nella organizzazione di un file, un bucket può essere un blocco o una sequenza contigua di blocchi.
- Record con stesso valore hash sono memorizzati nello stesso bucket/blocco



Organizzazione dei record all'interno del file: strutture primarie: accesso calcolato (hash)

Funzione hash applicate all'organizzazione di file

- Per gestire future collisioni, un blocco viene riempito solo parzialmente (fattore riempimento) in modo da lasciare spazio libero
- Dato
  - T - il numero di record previsti
  - F - il fattore di blocco
  - f - il fattore di riempimento (0,1)

  il file avrà un numero di blocchi $B = \lceil T/(f \times F) \rceil$

  Funzione hash applicata al campo chiave restituisce un valore da 0 a B-1

- Quando uno spazio relativo ad un blocco viene esaurito, viene allocato un ulteriore blocco collegato al precedente (catena di overflow)

Un esempio

- Matricola 6 cifre
- 42 record

- file hash con fattore di blocco 10;
- Servono 5 blocchi
- Funzione hash per indirizzare 5 blocchi
  - Mod 5

M
60600
66301
205751
205802
200902
116202
200604
66005
116455
200205
201159
205610
201260
102360
205460
205912
205762
200464
205617
205667

M
200268
205619
210522
205724
205977
205478
200430
210533
205887
200138
102338
102690
115541
206092
205693
205845
200296
205796
200498
206049

M'
220256
220258

Un file hash

file hash con fattore di blocco 10
5 blocchi con 10 posizioni ciascuno:
due soli overflow!

Numero medio di accessi alla struttura dati per un generico elemento? 1,05

60600	66301	205802	200268	200604
66005	205751	200902	205478	201159
116455	115541	116202	210533	200464
200205	200296	205912	200138	205619
205610	205796	205762	102338	205724
201260	220256	205617	205693	206049
102360		205667	200498	
205460		210522	220258	
200430		205977		
102690		205887		
205845		206092		

Un file hash

file hash con fattore di blocco 10
5 blocchi con 10 posizioni ciascuno:
due soli overflow!

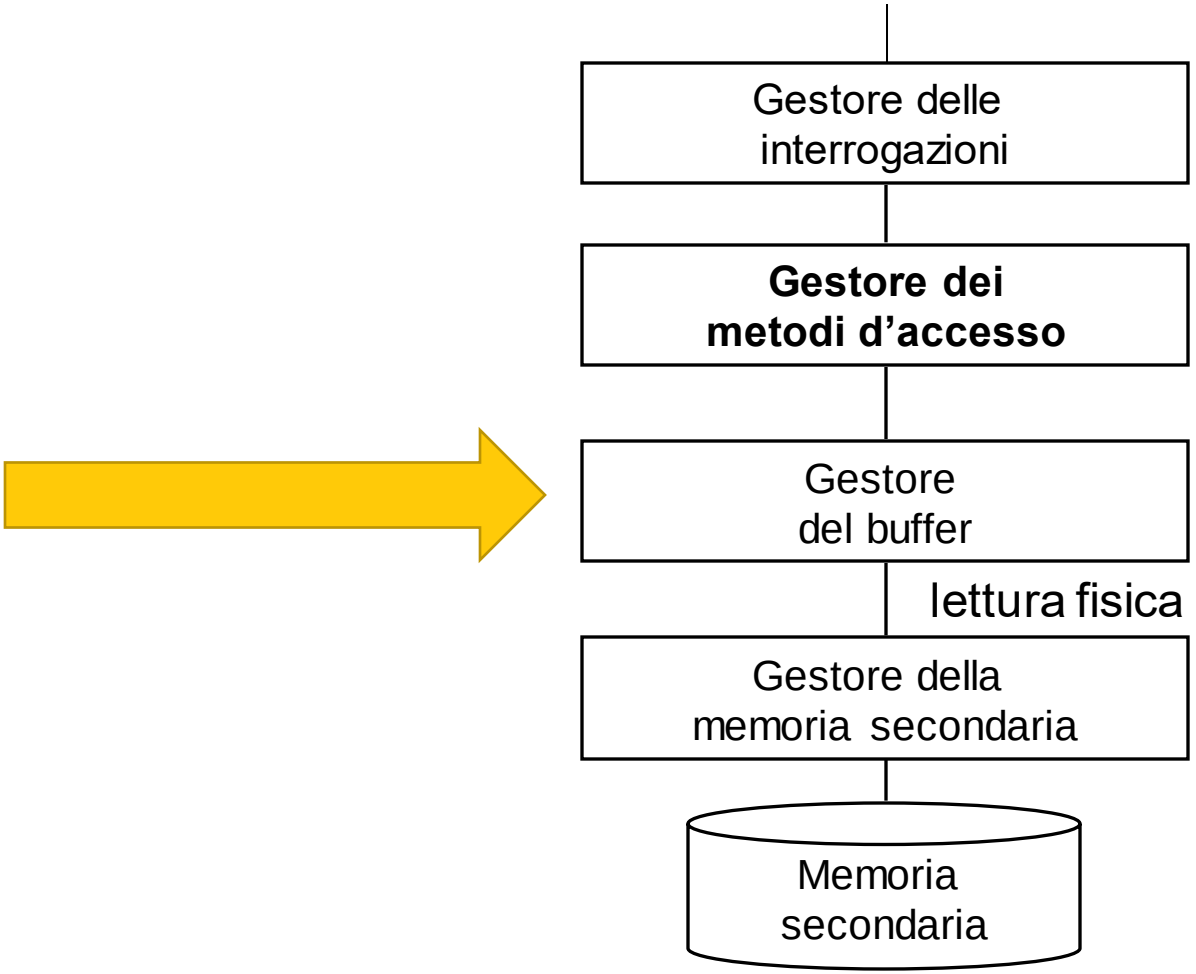
Numero medio di accessi alla struttura dati per un generico elemento? 1,1

60600	66301	205802	200268	200604
66005	205751	200902	205478	201159
116455	115541	116202	210533	200464
200205	200296	205912	200138	205619
205610	205796	205762	102338	205724
201260	220256	205617	205693	206049
102360		205667	200498	
205460		210522	220258	
200430		205977		
102690		205887		
205845		206092		

File hash, osservazioni

- È l'organizzazione più efficiente per l'accesso diretto basato su valori della chiave con condizioni di uguaglianza (accesso puntuale):
  - costo medio di poco superiore all'unità
- Non è efficiente per ricerche basate su intervalli (né per ricerche basate su altri attributi)
- Le collisioni (overflow) sono di solito gestite con blocchi collegati
- Non adatto per file che crescono tanto (oltre al fattore di riempimento)

Gestore degli accessi e delle interrogazioni



Buffer management

- **Buffer**:
  - area di memoria centrale, gestita dal DBMS (preallocata) e condivisa fra le **transazioni/programmi**
  - organizzato in **pagine** di dimensioni pari o multiple di quelle dei blocchi di memoria secondaria
  - è importantissimo per via della grande differenza di tempo di accesso fra memoria centrale e memoria secondaria

Scopo della gestione del buffer

- Ridurre il numero di accessi alla memoria secondaria
  - In caso di lettura, se la pagina è già presente nel buffer, non è necessario accedere alla memoria secondaria
  - In caso di scrittura, il gestore del buffer può decidere di differire la scrittura fisica
- Decidere quali pagine tenere nel buffer:
  - Massimizzare le probabilità che la pagina richiesta sia già nel buffer
  - Se necessario liberare pagine dal buffer, eliminare quelle che creano minor danno (aka che non saranno da ricaricare a breve)

Gestore del buffer

- Per gestire il buffer, il gestore mantiene un direttorio dove, per ogni pagina, mantiene
  - informazioni sul file fisico e il numero del blocco
  - due variabili di stato:
    - **Pin:** un contatore che indica quanti programmi utilizzano la pagina. Se il contatore è 0, la pagine è libera (unpinned).
    - **Dirty:** un bit che indica se la pagina è stata modificata

Funzioni del buffer manager

- Intuitivamente:
  - riceve richieste di lettura e scrittura (di pagine)
  - le esegue accedendo alla memoria secondaria solo quando indispensabile e utilizzando invece il buffer quando possibile
- Più formalmente, riceve richieste da transazioni attraverso:
  - ***fix***, ***unfix***, ***setDirty***, ***force***.

Interfaccia offerta dal buffer manager

- ***fix:*** richiesta di un blocco; richiede una lettura su memoria secondaria solo se il blocco non è già nel buffer. In questo caso, incrementa il contatore associato alla pagina.
- ***setDirty:*** comunica al buffer manager che la pagina è stata modificata
- ***unfix:*** indica che la transazione ha concluso l'utilizzo della pagina (decrementa il contatore associato alla pagina)
- ***force***: trasferisce in modo sincrono una pagina in memoria secondaria

Esecuzione della fix

Cerca il blocco nel buffer

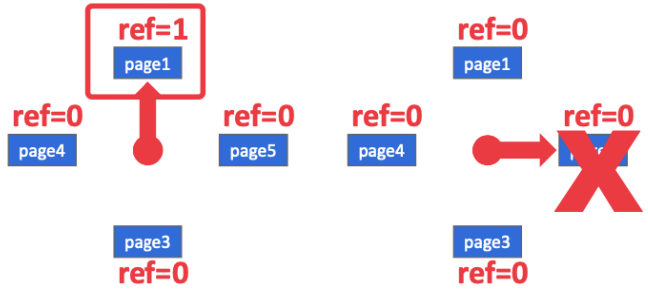
- se il blocco è già in una pagina del buffer,
  - incrementa il contatore e restituisce l'indirizzo della pagina
- altrimenti:
  - cerca le pagine unpinned nel buffer (contatore a zero);
    - tra queste ne sceglie una, seguendo diverse strategie (replacement policy);
    - se la pagina selezionata è stata modificata/dirty
      - viene aggiornata in memoria secondaria (flush)
    - restituisce l'indirizzo
  - Se non ci sono pagine unpinned, due alternative:
    - steal: seleziona una "vittima ", una pagina del buffer utilizzata (replacement policy);
      - I dati della vittima sono scritti in memoria secondaria (flush);
      - viene caricata la pagina di interesse dalla memoria secondaria e si restituisce l'indirizzo
    - no-steal: la transazione che ha richiesto la pagina viene posta in attesa

Replacement policy

- LRU- Least recently used.
  - Si scarica la pagina che è stata usata meno di recente. Questo richiede di memorizzare per ogni pagina l'orario dell'ultimo accesso.
    - ++ la prob. di accesso ad una pagina non utilizzata da tempo è bassa
- FIFO – first in first out.
  - Si scarica la pagina che è da più tempo nel buffer
    - -- Un blocco che rimane tanto tempo nel buffer, perchè viene tanto richiesto, potrebbe essere scartato per poi essere richieso e ricaricato a breve.

Replacement policy

- Clock replacement (politica dell'orologio)
  - Come LRU, ma si controllano le pagine in modo circolare (come round robin), tenendo in considerazione gli ultimi accessi alle pagine



- MRU - Most recent used
  - Si scarica la pagina che è stata utilizzata più recentemente (utile per alcuni tipi operazioni su base di dati (es. Join)

Replacement policy: Esempio

- Operazione di join Impiegati |x| Dipartimenti (assumendo che le relazioni siano in due file diversi)

Nested Loop Join

for each tuple I in IMPIEGATI **do**
  for each tuple D in $DIPARTIMENTO_s$ **do**
    if I.dip=D.ID
      add them in the result of the join
  end
end

- 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)

67

Replacement policy: Esempio

- Operazione di join Impiegati |x| Dipartimenti (assumendo che le relazioni siano in due file diversi)

Nested Loop Join

for each tuple I in IMPIEGATI **do**
  for each tuple D in $DIPARTIMENTO_s$ **do**
    if I.dip#=D.dip#
      add them in the result of the join
  end
end

- 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)

68

strategie: Force/No-force

- Il buffer manager richiede scritture in due contesti diversi:
  - in modo **sincrono** quando è richiesto esplicitamente con una force
  - in modo **asincrono** quando lo ritiene opportuno (o necessario, flush);

Strategia Force/no force:

- Force: tutte le pagine coinvolte da una transazione attiva vengono scritte in memoria di massa appena essa fa commit
- No-force: ci si affida al flush per scrivere la pagine di transazioni che hanno fatto commit

Altre ottimizzazioni

- Si può decidere di anticipare o posticipare scritture delle pagine (no-force policy) per coordinarle e/o sfruttare la disponibilità dei dispositivi:
  - Pre-flushing. Scaricamento anticipato delle pagine libere che sono state modificate nel corso del loro utilizzo (bit di stato con valore dirty).
  - pre-fetching. Si può anticipare anche i tempi di caricamento rispetto alle richieste delle transazioni, in quei casi in cui sono note a priori le modalità di accesso alle pagine della base di dati da parte di una transazione
- Osservazione: una pagina utilizzata da molte applicazioni può restare a lungo nel buffer, subendo varie modifiche, e venire trascritta in memoria secondaria con una sola operazione di scrittura.