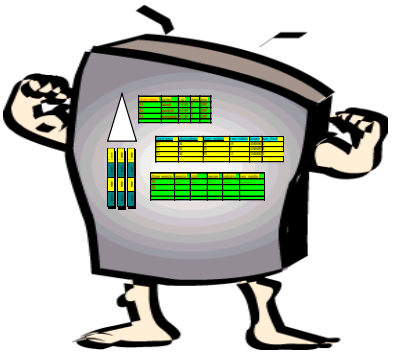
Organizzazioni dei dati – parte I



Dario Maio

http://bias.csr.unibo.it/maio/

Tipi di organizzazioni dei dati

- **Alcune diverse classificazioni d'interesse:**
- **Primaria vs secondaria:**
  - Un'organizzazione primaria, al contrario di un'organizzazione secondaria, impone un criterio di allocazione dei dati.
- **Statica vs dinamica:**
  - Un'organizzazione dinamica si adatta alla mole effettiva dei dati. Viceversa, un'organizzazione statica prevede fasi di "**riorganizzazione**" globale a fronte di variazioni, più o meno consistenti, del volume di dati da gestire.
- **Per chiave primaria vs chiave secondaria**
  - Il valore della chiave identifica un unico record in un'organizzazione per chiave primaria, e più record nel secondo caso.

- **NB:** *Il termine* **chiave** *indica una combinazione di campi che identifica univocamente un record; spesso è usato anche con il significato di* **chiave di ricerca**, *ovvero uno o più campi tramite i quali si accede ai dati.*

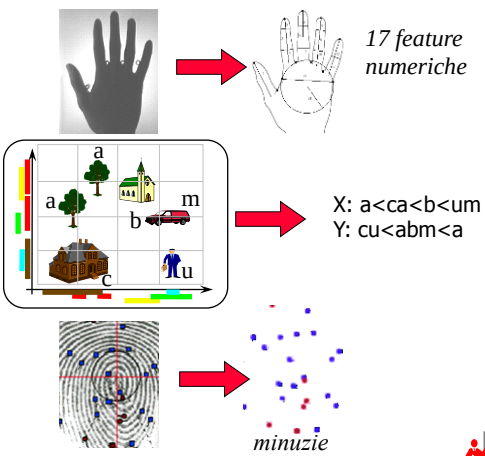
Tipi di operazioni

- **Le operazioni, per quanto complesse, sono riconducibili, in termini di I/O, ad alcune primitive di base:**
- **Ricerca** (*esatta*) che può essere per:
  - **chiave primaria: restituisce al più un solo record**
    es. lo studente con matricola 2106110234
  - **chiave secondaria: restituisce 0 o più record**
    es. gli studenti residenti a Cesena
  - **intervallo: restituisce 0 o più record**
    es. i contribuenti con reddito inferiore a 40.000 €
  - **varie combinazioni:**
    es. i giocatori dell'Inter o della Juventus di età inferiore a 20 anni
- **Inserimento di uno o più record**
- **Cancellazione di uno o più record**
- **Modifica di uno o più record**
  - **Può essere vista come la combinazione di cancellazione e inserimento.**

❖ **NB:** *Il termine* **transazione** *indica l'insieme di operazioni elementari che devono essere eseguite per soddisfare una determinata richiesta*

Alcune note (1)

- Si assume d'ora in avanti, per le operazioni di ricerca, di fare riferimento a ricerche di tipo esatto. In alcune applicazioni esiste invece la necessità di ricerche per similarità, ad esempio:
  - **Vector Space Model**: vettori di lunghezza predefinita d, sui quali è definita una metrica
  - **Metric Space Model**: elementi di uno spazio metrico generale (stringhe, grafi relazionali)
  - **Modelli generali**: la misura di similarità costruita sui modelli utilizzati per rappresentare gli oggetti non è una metrica

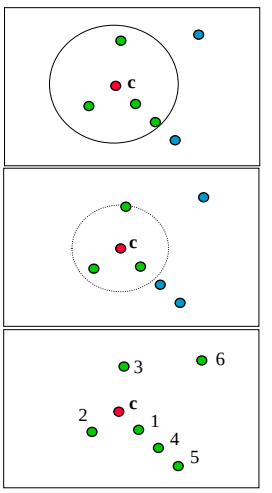


Alcune note (2)

- **Ricerche per similarità in uno spazio multidimensionale**

Dato un punto c, detto "**centro della query**":

- **Range query**: trovare tutti i punti distanti da c meno di una soglia fissata
- **k-nearest neighbor**: trovare i k punti più vicini a c
- **ricerche nearest neighbor incrementali** o **distance scan**: reperire incrementalmente i punti ordinati in base alla loro distanza da c



Clustering e Indexing (1)

- **Due concetti molto importanti per le organizzazioni dei dati**

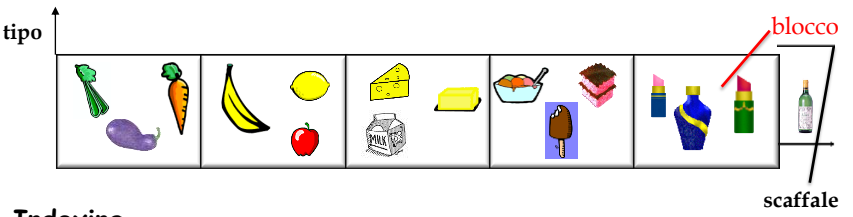
Clustering: presenza di "addensamenti" di dati, nei blocchi del file; si noti che l'ordinamento è un caso particolare.

La presenza di clustering è importante per ricerche su chiave secondaria, e può essere indotto da dipendenze esistenti tra gli attributi (si pensi, ad esempio, alla dipendenza tra età e stipendio di un impiegato, oppure fra tipo merce e scaffale di allocazione in un supermercato).

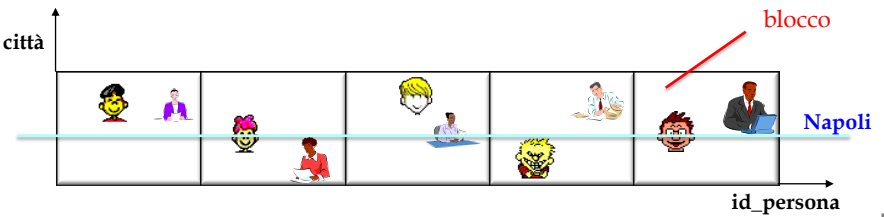
Indexing: a differenza del clustering che riguarda come i dati sono raggruppati nei blocchi, l'indexing concerne gli aspetti relativi all'accesso ai dati, cioè la possibilità effettiva di risolvere efficacemente operazioni di ricerca e aggiornamento.

Clustering e Indexing (2)

Clustering



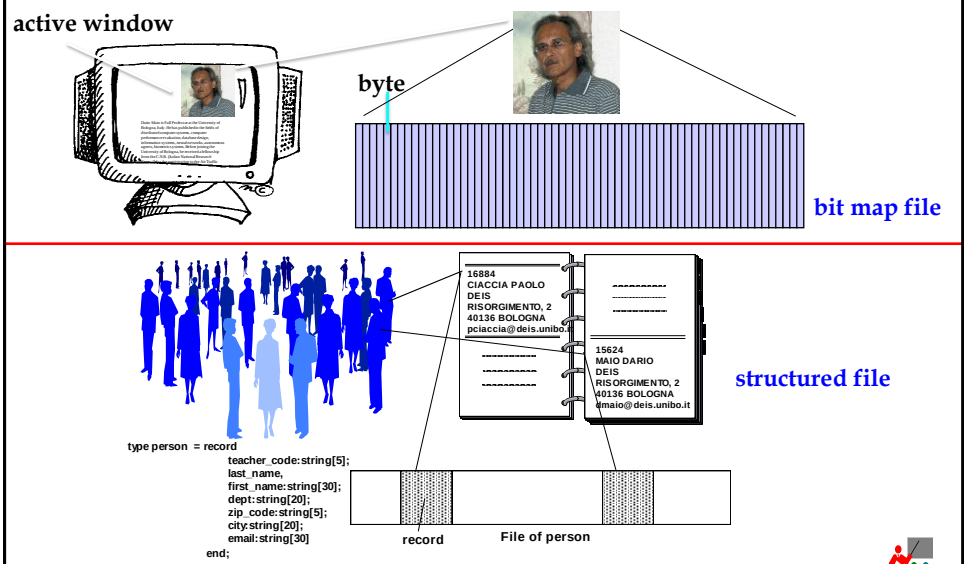
Indexing



Astrazioni di file (1)

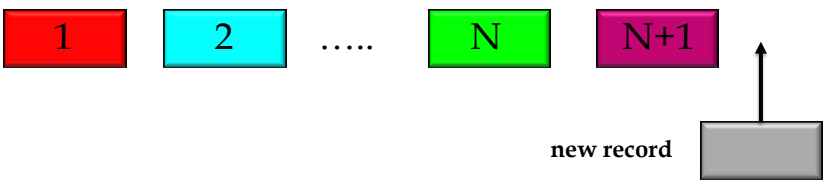
- A livello di applicazione un file è un'organizzazione di dati astratta. Una classificazione usuale ma discutibile…:
- **File "non strutturato"**: sequenza di byte (*stream*) su cui è possibile operare attraverso primitive orientate alla manipolazione del singolo byte o di blocchi di byte.
  - Esempi di uso sono: gestione di immagini bit map, pattern matching, gestione di dispositivi virtuali di I/O, operazioni che prescindono dalla struttura logica dei record (copie di backup, trasferimenti in rete, ecc).
- **File strutturato**: collezione di record, a lunghezza fissa o variabile, che appartengono a un certo tipo di dato astratto.
  - Esempi : un file di persone, un file di libri, un file di testo, …

Astrazioni di file (2)



Organizzazione sequenziale (1)

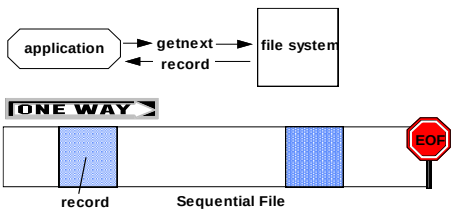
- L'organizzazione sequenziale è disponibile su nastri e dischi. Indipendentemente dalla reale allocazione fisica, la sequenza logica di registrazione può rispettare un ordinamento in base a un campo chiave o semplicemente l'ordine temporale della registrazione (in quest'ultimo caso l'organizzazione è detta ***seriale*** o ***heap***).



- La sequenza con cui i record possono venire letti è la stessa con la quale sono stati scritti sul file. Il record N-esimo è logicamente sempre tra il record (N-1)-esimo e il record (N+1)-esimo, non esiste spazio libero per inserimenti tra due record. L'inserimento di nuovi record avviene solo in fondo al file.

Organizzazione sequenziale (2)

- Per accedere in lettura a un record in un file sequenziale bisogna aprire il file (**OPEN**) e poi leggere tutti i record che precedono quello voluto. Il file system fornisce una primitiva "***get next***" per accedere al prossimo record in sequenza e una primitiva "***end_of_file***" per rilevare la fine del file.



- In generale, l'accesso sequenziale non permette di tornare al record i-esimo una volta letto l'(i+1)-esimo (**backspace**). Per tornare al record i si deve chiudere (**CLOSE**) il file, riaprirlo (**OPEN**) e rileggerlo fino al record i.
- N.B. *Spesso i file sequenziali, anche se su disco, sono equivalenti a file su nastro: i record non possono essere modificati e l'aggiornamento comporta una riscrittura del file. In alcuni file system è possibile modificare un record e reinserirlo se la lunghezza non è variata.*

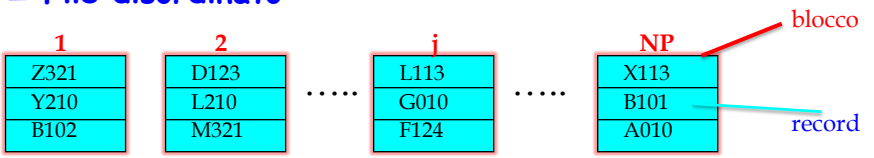
Utilità delle organizzazioni sequenziali

- Su disco sono utili in presenza di una o più delle seguenti situazioni:

1) Piccoli volumi di dati
2) Operazioni che interessano tutti o gran parte dei record, ad esempio:
   - ✓ lettura file di configurazione
   - ✓ calcolo della media dei valori di un campo
   - ✓ interrogazioni poco selettive.
3) Aggiornamenti non frequenti

Organizzazione sequenziale: ricerca per chiave primaria (1)

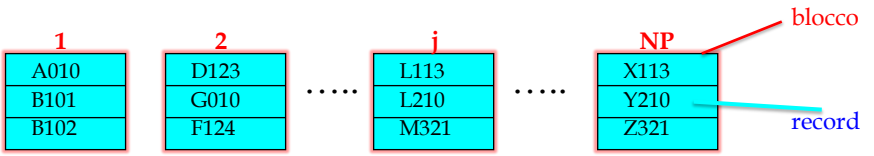
- **File disordinato**



- Se ogni blocco ha probabilità **1/NP** di ospitare il record cercato
- **caso di esistenza** del record cercato:
  - in media si accede a $\sum_{j=1}^{NP} j \times \frac{1}{NP} = \frac{NP+1}{2}$ blocchi;
  - nel caso peggiore si accede a NP blocchi.
- **caso di non esistenza** del record cercato:
  - si visitano NP blocchi.

Organizzazione sequenziale: ricerca per chiave primaria (2)

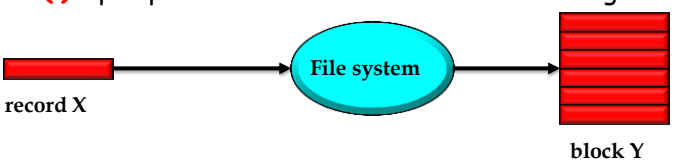
- **File ordinato sulla chiave primaria**



- Se ogni record ha la stessa probabilità di essere richiesto:
- **caso di esistenza** del record cercato:
  - in media si accede a (NP+1)/2 blocchi;
  - nel caso peggiore si accede a NP blocchi.
- **caso di non esistenza** del record cercato:
  - in media si accede a (NP+1)/2 blocchi; rispetto al caso disordinato si ha una diminuzione del costo di reperimento.

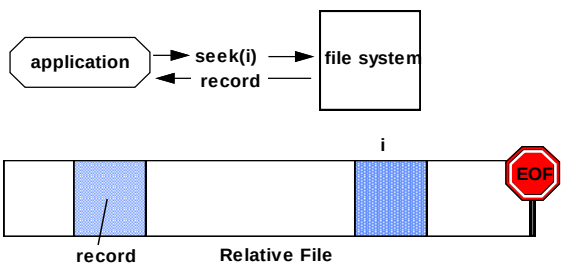
Organizzazione ad accesso diretto (1)

- Un'organizzazione ad accesso diretto (***relative***) consente di indirizzare ogni record tramite un numero, di solito da 0 a N-1 se N sono i record. A tale scopo il file system opera un mapping tra indirizzi logici di record e indirizzi di blocco (spesso si parla anche di bucket o pagina).
- Il tipo di dato astratto è fondamentalmente l'array di record, con dimensioni non prefissate a priori. Le primitive di base per l'accesso e la manipolazione sono:
  - ***get next*** : per proseguire, a partire da un certo indirizzo di record, nella sequenza logica degli indirizzi.
  - ***seek (i)*** : per posizionarsi sul record di indirizzo logico i.



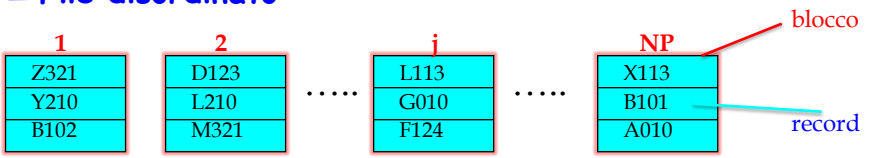
Organizzazione ad accesso diretto (2)

- Nella lettura sequenziale di un file relative, vengono considerati anche i record non ancora scritti. Ad esempio, avendo inserito solo i record di indirizzo logico 5 e 10 rispettivamente, posizionandosi sul record 5 e leggendo sequenzialmente i record da 5 a 10 si accede anche ai record 6,7,8,9 che hanno un contenuto impredicibile.
- N.B. *L'astrazione di file ad accesso diretto consente la costruzione di strutture d'accesso più complesse.*



File ad accesso diretto ricerca per chiave primaria (1)

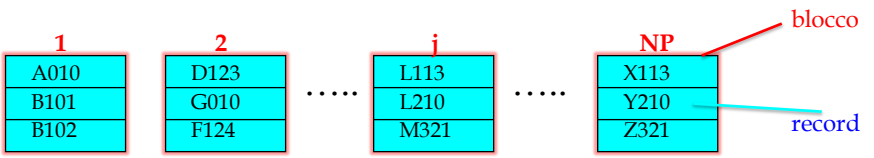
- File disordinato



- Se ogni blocco ha probabilità 1/NP di ospitare il record cercato
- **caso di esistenza** del record cercato:
  - in media si accede a $\sum_{j=1}^{NP} j \times \frac{1}{NP} = \frac{NP+1}{2}$ blocchi;
  - nel caso peggiore si accede a NP blocchi.
- **caso di non esistenza** del record cercato:
  - si visitano NP blocchi.

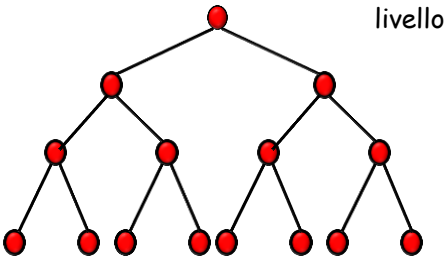
File ad accesso diretto: ricerca per chiave primaria (2)

- File ordinato sulla chiave primaria



- **Ricerca sequenziale** : sia in caso di esistenza sia in caso di non esistenza si accede in media a (NP+1)/2 blocchi;
- **Ricerca dicotomica**
  - **caso di esistenza** del record cercato (con NP>>1):
    - in media si accede a $\lfloor \log_2 NP \rfloor$ blocchi;
    - nel caso peggiore si accede a $\lfloor \log_2 NP \rfloor$ + 1 blocchi
  - **caso di non esistenza** del record cercato: si visitano al più $\lfloor \log_2 NP \rfloor$ + 1 blocchi

Note sulla ricerca dicotomica



Numero nodi $n = 2^{h+1} - 1$

LIVELLO	COSTO	N° CASI
0	1	1
1	2	2
2	3	4
3	4	8
….		
h	h+1	2^h

caso di esistenza, in media si accede a un numero di blocchi pari a:

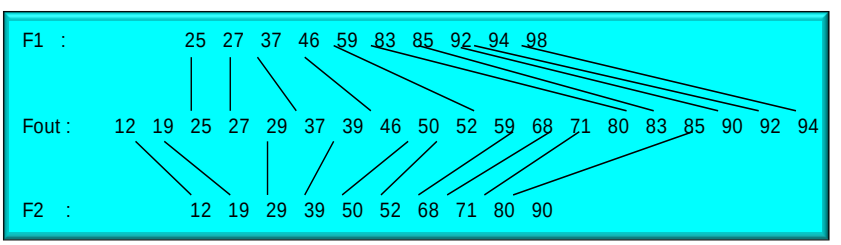
$$\frac{\sum_{i=1}^{h+1} i \cdot 2^{i-1}}{2^{h+1} - 1} \cong \lfloor \log_2 n \rfloor = h$$

caso peggiore

$$\lfloor \log_2 n \rfloor + 1 = h + 1$$

Fusione di archivi ordinati

- Tra le prime operazioni per cui sono stati studiati algoritmi efficienti che tenessero in debita considerazione le caratteristiche dei dispositivi di memoria secondaria si trovano la fusione (***merge***) e l'ordinamento (***sort***) di archivi memorizzati su file.
- L'operazione di fusione di due o più archivi presuppone che sia gli archivi in input sia l'archivio in output siano ordinati secondo un criterio comune.



Algoritmo di fusione di F File

1. **Inizializzazione**

   Si legge il primo record da ognuno degli F file e si inserisce nell'insieme dei record correnti ***RC***.

   repeat

   2. **selezione**

      Si scrive in output il record con il più piccolo valore di chiave, tra quelli in ***RC***, e si elimina da ***RC***.

   3. **rimpiazzamento**

      Si legge un record, se esiste, dal file da cui è stato scelto il record al passo di selezione e si inserisce in ***RC***.

   until ***RC*** = ∅;

Ordinamento di archivi (1)

attributi nel risultato

attributi di ordinamento

Matricola	CodiceFiscale	Cognome	Nome	DataNascita
210629323	BNCGRG78L21A944K	Bianchi	Giorgio	21/07/1978
216635467	RSSNNA78A53A944V	Rossi	Anna	13/01/1978
160239654	VRDMRC79H21F839X	Verdoni	Marco	21/06/1978
214842132	VRDCRL79H20G125J	Verdi	Carlo	20/06/1979
2006431216	RSSDRA78M10A944V	Rossi	Dario	10/08/1978

Matricola	Cognome	Nome	DataNascita
210629323	Bianchi	Giorgio	21/07/1978
216635467	Rossi	Anna	13/01/1978
2006431216	Rossi	Dario	10/08/1978
214842132	Verdi	Carlo	20/06/1979
160239654	Verdoni	Marco	21/06/1978

Ordinamento di archivi (2)

- L'interesse per gli algoritmi di fusione nasce dal fatto che essi intervengono nell'operazione di **ordinamento esterno**, secondo il seguente schema generale:

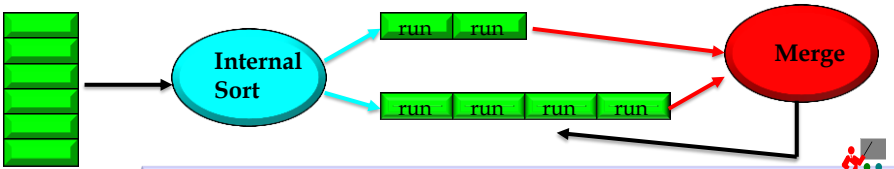
1. **Sort interno**

   Dato un archivio di N record, e uno spazio in memoria centrale che possa ospitare $M < N$ record, si possono ordinare *internamente* i record a gruppi di M, producendo $\lceil N/M \rceil$ *sotto-archivi*, detti anche *sequenze* o *run*.

2. **Merge**

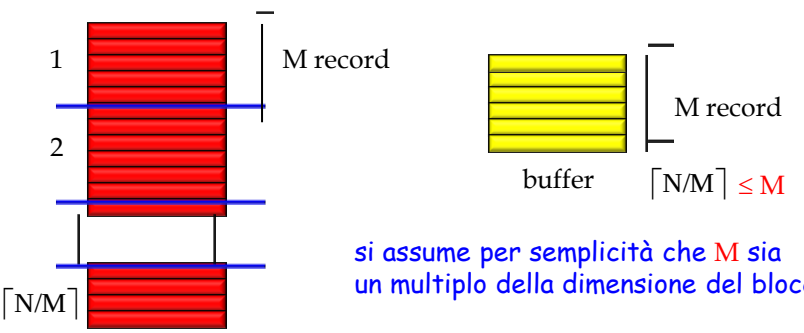
   Le $\lceil N/M \rceil$ run vengono fuse producendo un singolo archivio ordinato.

- **Nella fase 1 si può far ricorso a uno qualsiasi degli algoritmi di sort interno** (es.: insertion, quicksort, heapsort, mergesort, ..).
- **La fase 2 può richiedere più passi** e diversificarsi a seconda del tipo di distribuzione delle run sui file ausiliari utilizzati dall'algoritmo.



Sort-Merge orientato ai record

- Questo algoritmo è adeguato per archivi di modeste dimensioni, e l'ordinamento è gestito a livello di applicazione, avendo come sola astrazione l'organizzazione relative.
- **Caso particolare: $N \leq M^2$**
- Il numero N di record, memorizzati in un file di NP blocchi, soddisfa la relazione $M < N \leq M^2$, essendo M la capacità in record dell'area dati in memoria centrale a disposizione dell'applicazione.

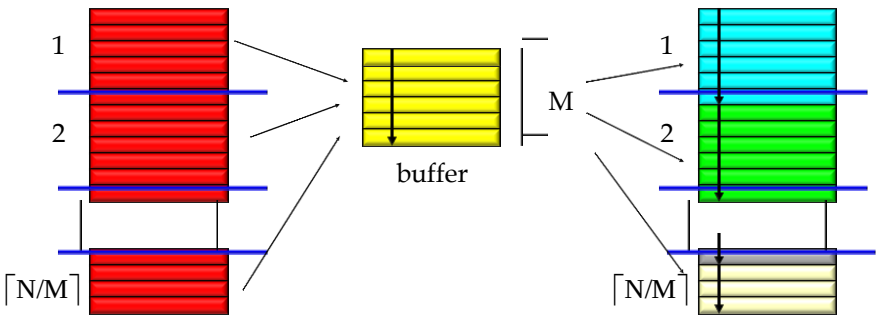


si assume per semplicità che M sia un multiplo della dimensione del blocco

Sort-Merge: caso N ≤ M² (1)

- **Sort interno**

  Si ordinano internamente, utilizzando il buffer a disposizione, i record a gruppi di M, producendo $\lceil N/M \rceil \leq M$ sequenze ordinate.



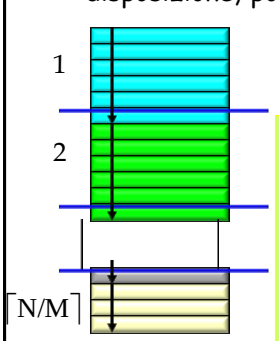
file origine

file temporaneo

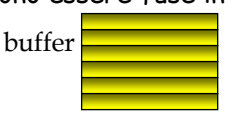
Sort-Merge: caso N ≤ M² (2)

- **Merge**

  Poiché le sequenze sono in numero <=M, utilizzando ancora il buffer a disposizione, possono essere fuse in un unico passo.



file temporaneo



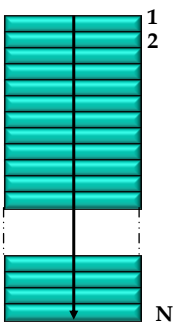
Si legge il primo record da ognuna delle ⌈N/M⌉ sequenze e si inserisce nel buffer.

repeat

selezione: si scrive sul file destinazione il record con il più piccolo valore di chiave, tra quelli nel buffer, e si elimina dal buffer.

rimpiazzamento: si legge un record, se esiste, dalla sequenza da cui è stato scelto il record al passo di selezione e si inserisce nel buffer.

until buffer vuoto;



file destinazione

Sort-Merge: caso $N \leq M^2$ (3)

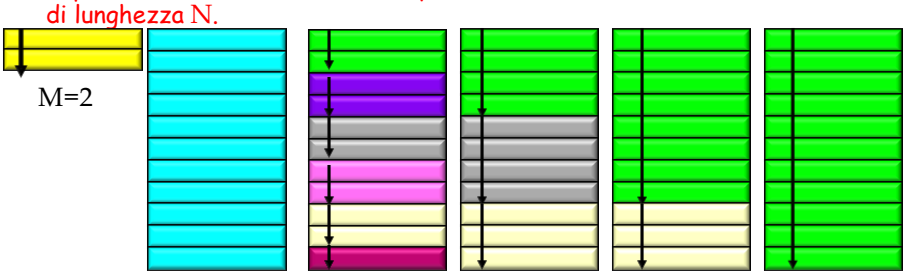
- **Complessità dell'algoritmo**
  - **sort interno**
    si leggono NP blocchi e si scrivono NP blocchi
  - **passo di merge**
    assumendo un **hit ratio** pari a 0 (caso peggiore), ogni lettura di record comporta l'accesso a un blocco; pertanto si leggono N blocchi e si riscrivono NP blocchi in sequenza.

- **Complessivamente, il costo in termini di I/O è pari a:**

$$C(\text{record Sort-Merge}) = N + 3 \times NP \qquad (N \leq M^2)$$

Sort-Merge caso $N > M^2$ (1)

- Si rendono necessari più passi di fusione, in quanto il numero di run generate dal sort interno è in questo caso superiore a M.
- L'input del primo passo di fusione è dato da $\lceil N/M \rceil$ run di lunghezza M (a parte, eventualmente, l'ultima). Si possono fondere M run alla volta, ottenendo pertanto $\lceil N/M^2 \rceil$ run di lunghezza M^2.
- Al secondo passo le run in input hanno lunghezza M^2; fondendole M alla volta si ottengono run $\lceil N/M^3 \rceil$ di lunghezza M^3.
- Il processo di fusione si arresta quando si ottiene una sola run ordinata di lunghezza N.



Sort-Merge: caso N > M² (2)

- **Complessità dell'algoritmo**
  - **sort interno**

    si leggono NP blocchi e si scrivono NP blocchi
  - **passi di merge**

    All'ultimo passo di fusione, PF, si genera una sola run di lunghezza $N \le M^{PF+1}$, pertanto:

$$PF = \left\lceil \log_M \frac{N}{M} \right\rceil = \lceil \log_M N \rceil - 1$$

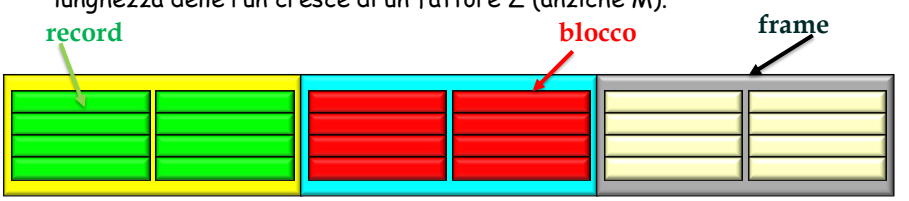
- **Complessivamente, il costo di I/O è pari a:**

$$C(\text{record Sort} - \text{Merge}) = 2 \times NP + (N + NP) \times \left\lceil \log_M \frac{N}{M} \right\rceil$$

- La dipendenza da N evidenzia che l'algoritmo non sfrutta adeguatamente l'organizzazione a blocchi dei record.

Sort-Merge a Z vie (1)

- L'inefficienza del record Sort-Merge dipende dal fatto che i record considerati, a ogni passo elementare, per la fusione, appartengono a M run distinte. Questo fa sì che il costo di lettura, per ogni passo di fusione, sia proporzionale a N.
- L'algoritmo Sort-Merge a Z vie organizza l'area di memoria centrale in Z pagine logiche (**frame**), ognuna di capacità M/Z record. Ogni frame corrisponde a FS ≥ 1 blocchi del file. La fusione è eseguita considerando Z run alla volta.
- Il costo di lettura di ogni passo di fusione risulta proporzionale al numero di blocchi, NP, del file, in quanto ogni blocco viene letto una sola volta. Per contro, il numero di passi di fusione aumenta, in quanto a ogni passo la lunghezza delle run cresce di un fattore Z (anziché M).



Esempio: Z=3, FS=2, M=24

Sort-Merge a Z vie (2)

- L'algoritmo fu inizialmente sviluppato per memorie a nastri. In questo caso (così come nel caso di organizzazioni sequenziali su disco) si rendono necessari $2 \times Z$ dispositivi, ognuno dei quali memorizza un file (sequenziale) ausiliario. Nel caso di organizzazioni relative, è sufficiente un singolo file ausiliario, dove vengono opportunamente memorizzate tutte le run. **Per semplicità, si suppone di usare $2 \times Z$ file ausiliari.**
- **1. Sort interno**: si ordinano internamente, utilizzando l'area dati a disposizione, i record a gruppi di M, producendo $\lceil N/M \rceil$ run. Il numero di run iniziali è anche esprimibile come $\lceil NP/(Z \times FS) \rceil$, ragionando in termini di blocchi, e la lunghezza di ciascuna run è pari a $Z \times FS$ blocchi (= Z frame). Le run vengono scritte **ciclicamente** sui primi Z file ausiliari. I file ausiliari si scambiano ruolo a ogni passo di fusione.
- **2. Passi di merge**: si fondono Z run alla volta, ognuna allocata in un frame. Dopo il primo passo di fusione si avranno $\lceil NP/(Z^2 \times FS) \rceil$ run di lunghezza Z^2 x FS blocchi (= Z^2 frame), che sono ciclicamente scritte sui secondi Z file ausiliari. All'ultimo passo di fusione PF si ha una sola run ordinata di lunghezza $NP \leq Z^{PF+1} \times FS$ blocchi.

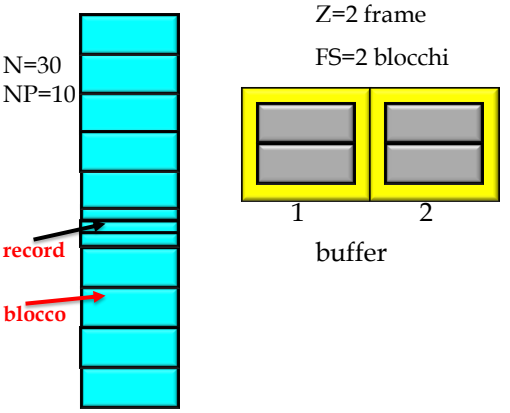
Sort-Merge a Z vie: esempio (1)

- **Sort interno**

M=12 record

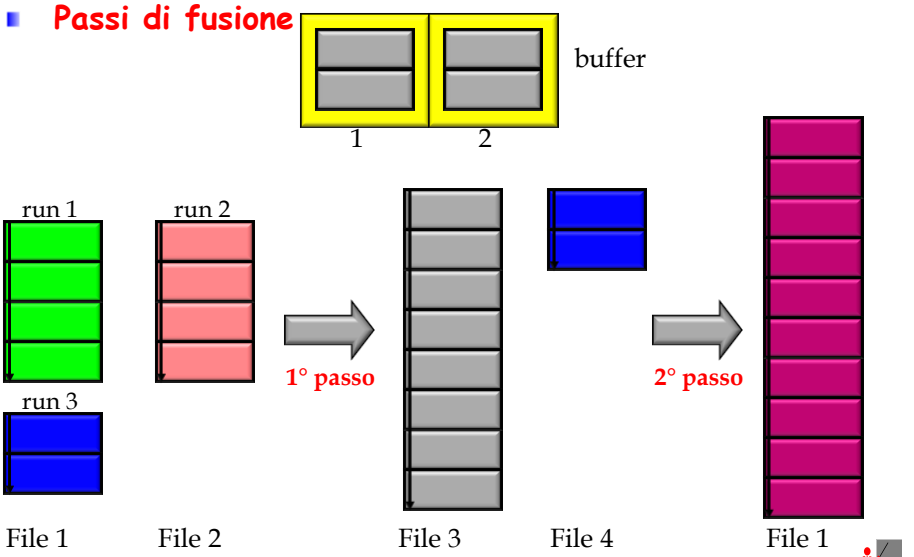
Z=2 frame

FS=2 blocchi



Sort-Merge a Z vie: esempio (2)

- **Passi di fusione**



Sort-Merge a Z vie: complessità

- **Complessità dell'algoritmo**
  - **sort interno**
    si leggono NP blocchi e si scrivono NP blocchi
  - **passi di merge**
    all'ultimo passo di fusione, PF, si ottiene una sola run di lunghezza $NP \leq Z^{PF+1} \times FS$ blocchi, pertanto:

$$PF = \left\lceil \log_z \frac{NP}{Z \times FS} \right\rceil = \left\lceil \log_z \frac{N}{M} \right\rceil$$

    A ogni passo di fusione si leggono e si scrivono NP blocchi.

- Complessivamente, il costo in termini di I/O è pari a:

$$\text{C(Z-way Sort-Merge)} = 2 \times NP \times \left(1 + \left\lceil \log_z \frac{NP}{Z \times FS} \right\rceil \right) = 2 \times NP \times \left\lceil \log_z \frac{NP}{FS} \right\rceil$$

Scelta della dimensione del frame

- A parità di **M** è preferibile **Z** elevato, **FS** basso o **Z** basso e **FS** elevato?
  - **Poiché il numero di passi di fusione è pari a** $\left\lceil \log_Z \frac{N}{M} \right\rceil$ conviene massimizzare Z, scegliendo FS=1
    o comunque, in caso di limite al numero 2×Z di file ausiliari che si possono tenere aperti, per organizzazioni sequenziali, è comunque vantaggioso massimizzare Z.
  - Posto **FS=1**, il costo dell'algoritmo diventa:

    C(Z-way Sort-Merge)= $2 \times NP \times \lceil \log_Z NP \rceil$
  - Si noti che nel caso di blocchi di capacità unitaria (N=NP) e FS=1, e dunque Z=M, l'algoritmo si riduce a quello orientato ai record; infatti:

    $2 \times NP + (N + NP) \times \left\lceil \log_M \frac{N}{M} \right\rceil = 2 \times NP \times (1 + \lceil \log_Z NP \rceil - 1)$

Esempio

- **File con N=NP=16 record, spazio in memoria centrale di Z=3 blocchi (FS=1).**
- **Il numero dei passi di fusione è** $\left\lceil \log_3 \frac{16}{3} \right\rceil = 2$

3	5	10	2	20	1	4	7	12	11	18	6	30	17	9	14

- **Sort interno: si producono 6 run di lunghezza 3 distribuite sui primi 3 file ausiliari**

F1:	3	5	10	6	11	18
F2:	1	2	20	9	17	30
F3:	4	7	12	14		

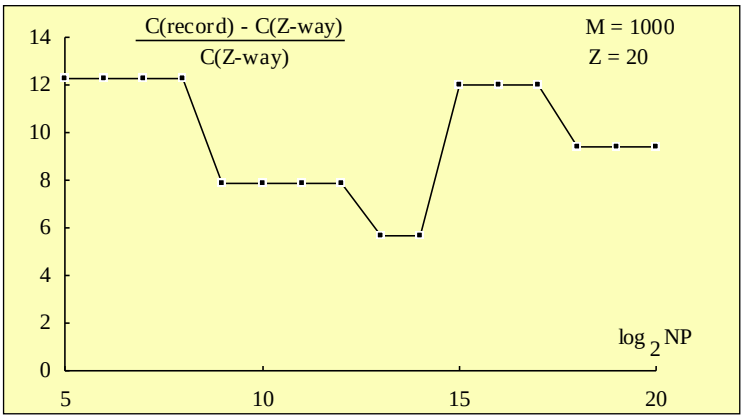
- **I passo di merge: 2 run di lunghezza massima 9 distribuite su altri 2 file**

F4:	1	2	3	4	5	7	10	12	20
F5:	6	9	11	14	17	18	30		

- **II passo di merge: 1 run di lunghezza 16, fondendo le due run di F4 e F5**

Record S-M vs Z-way S-M (1)

- Con blocchi di capacità (in record) > (>>) 1 e FS=1, l'algoritmo Z-way è chiaramente conveniente. Ad esempio per N=50 K record (NP=1K), il costo di ordinamento misurato in numero di operazioni di I/O è rispettivamente 54272 adottando record sort-merge e 6144 adottando Z-way.



Record S-M vs Z-way S-M (1)

- Il numero dei passi di fusione dell'algoritmo Z-way è maggiore o uguale a quello dell'algoritmo Record Sort-Merge.