Capitolo 11

Esercizio 11.1

Definire le strutture dati necessarie per la gestione dei buffer; implementare in un qualunque linguaggio le funzioni `fix`, `use` e `unfix`. Si suppongano disponibili le funzioni di file system descritte nel paragrafo 11.1.3.

Soluzione:

```
typedef struct page {
      int *address;
      int valid;
      int modified;
      int in_use;
      int file_id;
      int block_num;
}

typedef struct file_open {
      int file_id;
      char *file_name;
      int size;
      int blocks[]; // each element refers to an element in page table
}

typedef page_table *page;
typedef file_table *file_open;

page_table pt=new page_table[N];
file_table ft=new file_table[M];


int fix(char *file_name, int block) {
      int i=0;
      int found=0;
      while (i<0 && !found)
        if (strcmp(file_name,ft[i].file_name)!=0) i++; else found=1;

      if (found && ((ft[i].blocks[block])!=-1) &&
           pt[ft[i].blocks[block]].valid )
                      // -1 means that the block is
                      // not loaded
       return (pt[ft[i].blocks[block]]).address;

      if (!found) { // file is open
           int id=open(file_name);
           int position=get_file_position() // find a free position in ft
           ft[position].file_id=id;
           strcpy(file_name, ft[position].file_name);
           ft[position].size=get_size(id);
           for(int k=0; k<ft[position].size; k++)
                ft[position].blocks[k]=-1;
      }

      // block is read

      int free=get_free_page(); // find a free page in pt
```

```
      read(ft[i].file_id, block, free.address);
      pt[free].valid=1;
      pt[free].modified=0;
      pt[free].file_id=ft[i].file_id
      pt[free].block_num=block;
      return pt[free].address;
}

void use (int *page) {
      int i=0;
      int found=0;
      while (i<N && !found)
      if (pt[i].address!=page) i++ else found=1;
            pt[i].in_use=1;
}

void unfix(int *page) {
      int i=0;
      int found=0;
      while (i<N && !found)
            if (pt[i].address!=page) i++ else found=1;
      pt[i].in_use=0;
      if (pt[i].modified) {
            pt[i].valid=0;
            int j=0;
            found=0;
            while (j<N && !found)
                  if (ft[j].file_id!=pt[i].file_id) j++ else found=1;
            ft[j].block[pt[i].block_number]=-1;
      }
}
```

Esercizio 11.2

Si consideri una base di dati gestita tramite hashing, il cui campo chiave contenga i seguenti nominativi:
Green, Lovano, Osby, Peterson, Pullen Scofield, Allen, Haden, Sheep, Harris, MacCann, Mann, Brown, Hutcherson, Newmann, Ponty, Cobbham, Coleman, Mingus, Lloyd, Tyner, Fortune, Coltrane.

1. Proporre un algoritmo di hashing con B=8 e F=4.
2. Supponendo B=40 e F=1, qual è la probabilità di conflitto? E con B=20 e F=2?
3. Con F=5 e B=7, quanto vale approssimativamente la lunghezza media della catena di overflow?

Soluzione:

1) Una semplice funzione di hashing per i nomi dati è:
   - Per ogni carattere del nome, considerare il corrispondente numero in ordine alfabetico (a = 1, b = 2 …)
   - Sommare tutti i numeri ottenuti e fare il "modulo B" della divisione

In questo caso otterremo per ogni nome un numero compreso tra 0 e B-1.

Esempio:

Hash(Green)=(7+18+5+5+14) mod 8=1
Hash(Lovano)=(12+15+22+1+13+15) mod 8=6
Hash(Osby)=(15+19+2+25) mod 8=5
Hash(Peterson)=(16+5++20+5+18+19+15+13) mod 8=7

2) Con B = 40 e F = 1 la probabilità di conflitto è:

$$p = 1 - T\,(1/B)\,(1 - 1/B)^{T-1} = 1 - 23\,(1/40)\,(39/40)^{22} = 0{,}6706$$

Con B = 20 e F = 2

$$p = 1 - \sum_{i=1}^{F} \binom{T}{i} (1/B)^i (1-1/B)^{T-1} = 1 - 23(1/20)(19/20)^{22} - \binom{23}{2}(1/20)^2(19/20)^{21} = 0{,}4311$$

Nota che questa è la probabilità di avere due o più collisioni nello stesso blocco, perché ogni blocco contiene 2 tuple e 1 collisione è ammessa.

3) La lunghezza della catena di overflow può essere calcolata come la somma pesata delle probabilità di collisione.

$$l = \sum_{i=F+1}^{T} i \binom{T}{i} (1/B)^i (1-1/B)^{T-1} = 0{,}647$$

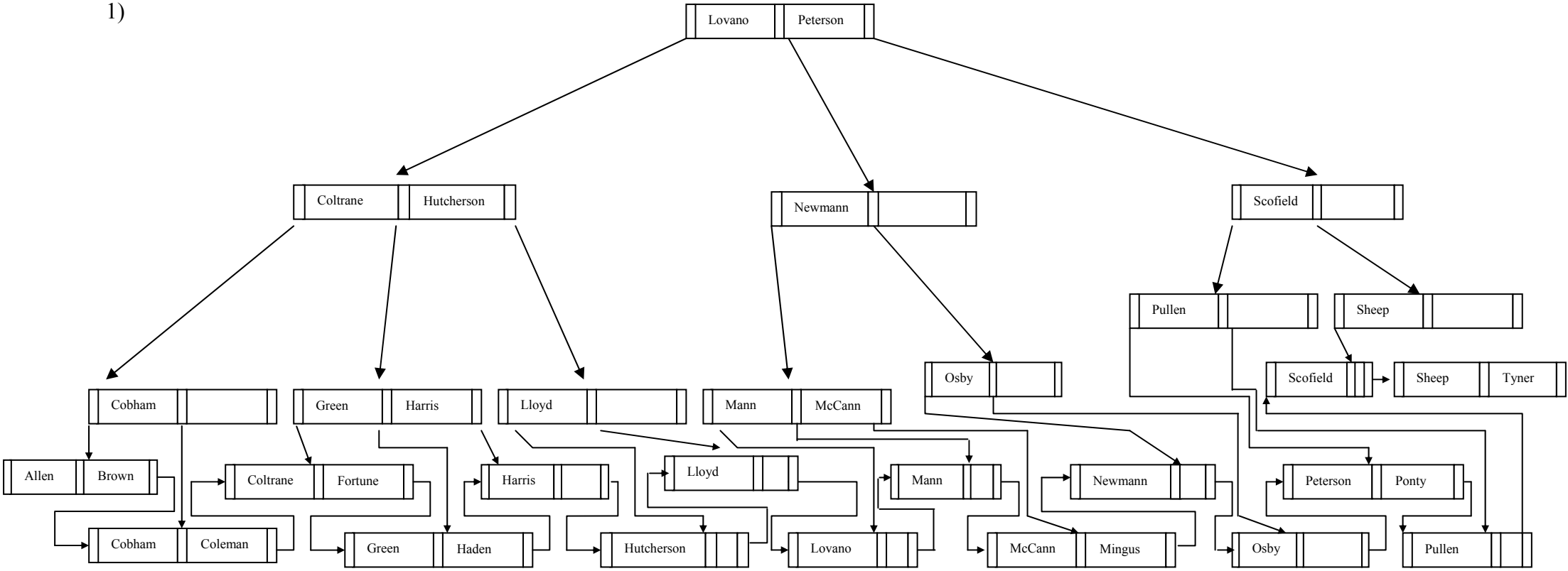
Esercizio 11.3

Si consideri una base di dati gestita tramite alberi B+, il cui campo chiave contenga i dati elencati nel precedente esercizio.

1. Descrivere una struttura ad albero B+ bilanciato, con F=2, che contenga i dati citati
2. Introdurre un dato che provochi lo split di un nodo al livello foglia, e mostrare cosa accade al livello foglia e al livello superiore.
3. Introdurre un dato che provochi il merge di un nodo al livello foglia, e mostrare cosa accade al livello foglia e al livello superiore.
4. Indicare una sequenza di inserimenti che provochi, ricorsivamente, lo split della radice e l'allungamento dell'albero.
5. Descrivere una struttura ad albero B, con F=3, che contenga i dati citati.

Soluzione:

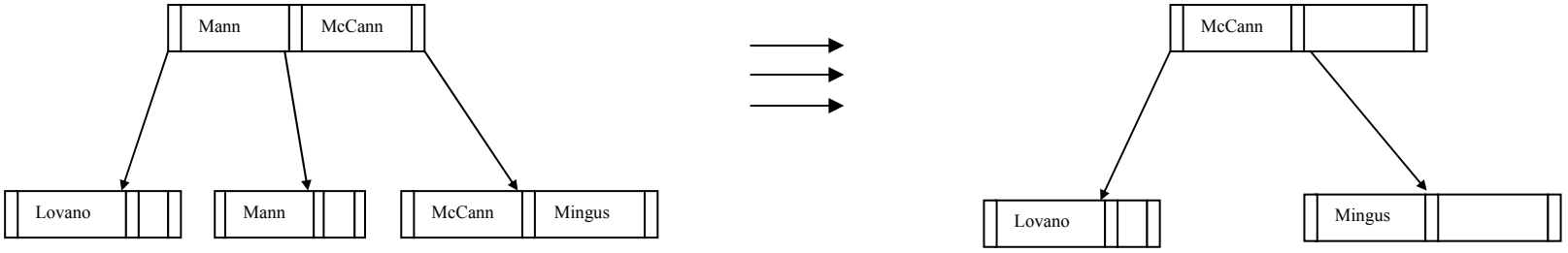
1)



L'albero B+Tree contiene tutti i campi chiave. Ha 16 nodi al livello foglia.

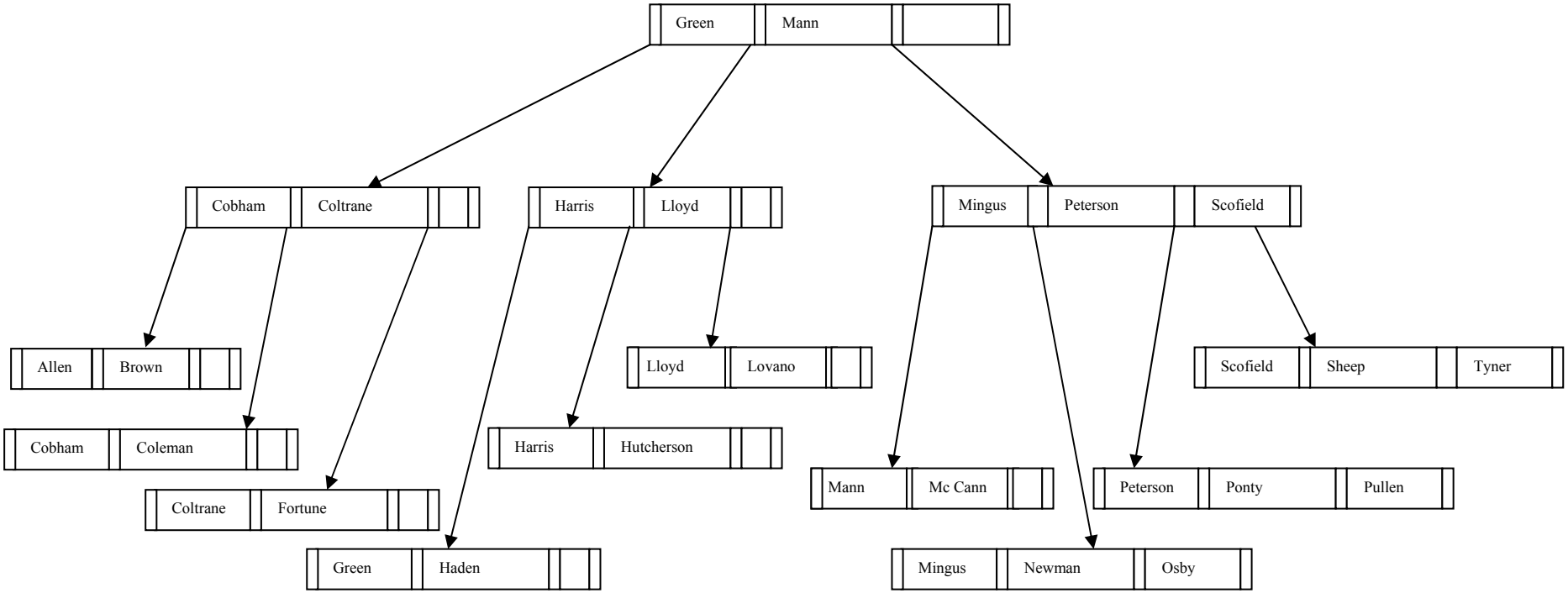
2) L'introduzione del valore "Brooke", causa uno split a livello foglia, come illustrato nella seguente figura.

3) La cancellazione del valore "Mann" causa un merge al livello foglia.



4) L'inserimento di "Brooke", "Addams" e "Gibson" causa uno slit alla root.

5)



Esercizio 11.4

Si consideri una base di dati costituita dalle seguenti relazioni:

PRODUZIONE(NumeroSerie, TipoParte, Modello, Quan, Macchina)
PRELIEVO(NumeroSerie, Lotto)
ORDINE(Lotto, Cliente, Ammontare)
COMMISSIONE(Lotto. Venditore, Ammontare)

Si assumano inoltre i seguenti profili:

CARD(PRODUZIONE) = 200.000	SIZE(PRODUZIONE)=41
CARD(PRELIEVO)=50.000	SIZE(PRELIEVO)=15
CARD(ORDINE)=10.000	SIZE(ORDINE)=45
CARD(COMMISSIONE)=5.000	SIZE(COMMISSIONE)=35

SIZE(NumeroSerie)=10	VAL(NumeroSerie)=200.000
SIZE(TipoParte)=1	VAL(TipoParte)=4
SIZE(Modello)=10	VAL(Modello)=400
SIZE(Quan)=10	VAL(Quan)=100
SIZE(Macchina)=10	VAL(Macchina)=50
SIZE(Lotto)=5	VAL(Lotto)=10.000
SIZE(Cliente)=30	VAL(Cliente)=400
SIZE(Ammontare)=10	VAL(Ammontare)=5.000
SIZE(Venditore)=20	VAL(Venditore)=25

Descrivere l'ottimizzazione algebrica e il calcolo dei profili dei risultati intermedi relativi alle seguenti interrogazioni, che vanno inizialmente espresse in SQL e poi tradotte in algebra relazionale:

1. Determinare la quantità disponibile del prodotto 77Y6878.
2. Determinare le macchine utilizzate per la produzione dei pezzi venduti al cliente Rossi.
3. Determinare i clienti che hanno comprato dal rivenditore Bianchi un box modello 3478.

Per le ultime interrogazioni, che richiedono join fra tre o quattro tabelle, indicare gli ordinamenti tra join che sembrano più convenienti sulla base delle dimensioni degli operandi. Descrivere poi, prevedendo solo due alternative a scelta per l'esecuzione dei join, l'albero delle alternative relativo alla seconda interrogazione.

Soluzione:

1) SQL:

```
select Quan
from Produzione
where NumeroSerie="77Y6878"
```

Algebra Relazionale:

$$\Pi_{Quan}(\sigma_{NumeroSerie=\text{"77Y6878"}}(Produzione))$$

Questa query non ha bisogno di nessuna ottimizzazione algebrica. Se indichiamo con T la tabella dei risultati, il profilo è:

CARD(T)=1 SIZE(Quan)=10
SIZE(T)=10 VAL(Quan)=1

2) SQL:

```
select Macchina
from Produzione join Prelievo on
Produzione.NumeroSerie=Prelievo.NumeroSerie
join Ordine on Prelievo.Lotto=Ordine.Lotto
where Cliente= ”Rossi”
```

Algebra relazionale:

$$\Pi_{Macchina}(\sigma_{Cliente=\text{"Rossi'}}(Produzione \bowtie_{NumeroSerie=p} \rho_{o,p \leftarrow NumeroSerie,\ Lotto}(Prelievo) \bowtie_{o=Lotto} Ordine))$$

Ottimizzazione algebrica: Push delle proiezioni e delle selezioni

$$\Pi_{Macchina}(\Pi_{Pr}(\Pi_{NumeroSerie}(\sigma_{Cliente=\text{"Rossi'}}(Ordine)) \bowtie_{NumeroSerie=Or} \rho_{Or,Pr \leftarrow NumeroSerie,\ Lotto}(Prelievo)) \bowtie_{Pr=Lotto} (\Pi_{Lotto,\ Macchina}(Produzione)))$$

Poniamo: $T_1 = \Pi_{NumeroSerie}(\sigma_{Cliente=\text{"Rossi'}}(Ordine))$

Abbiamo:

$CARD(T_1) = (1 / VAL(Cliente)) * CARD(Ordine) = (1 / 400) * 10.000 = 25$

$SIZE(T_1) = 5$

La scelta più conveniente in ordine di join è $(Ordine \bowtie Prelievo) \bowtie Produzione$

Poniamo: $T_2 = \Pi_{Pr}(T_1 \bowtie OrderDetail)$

$CARD(T_2) = CARD(T_1) * CARD(Prelievo) * (1 / VAL(Lotto)) = 25 * 50.000 * (1 / 10.000) = 125$

$SIZE(T_2) = 10$

Notare che la proiezione può essere eseguita con lo scan, in questo modo non sono necessari altri risultati intermedi.

Poniamo: $T_3 = \Pi_{NumeroSerie,\ Macchina}(Produzione)$

$CARD(T_3) = 200.000$

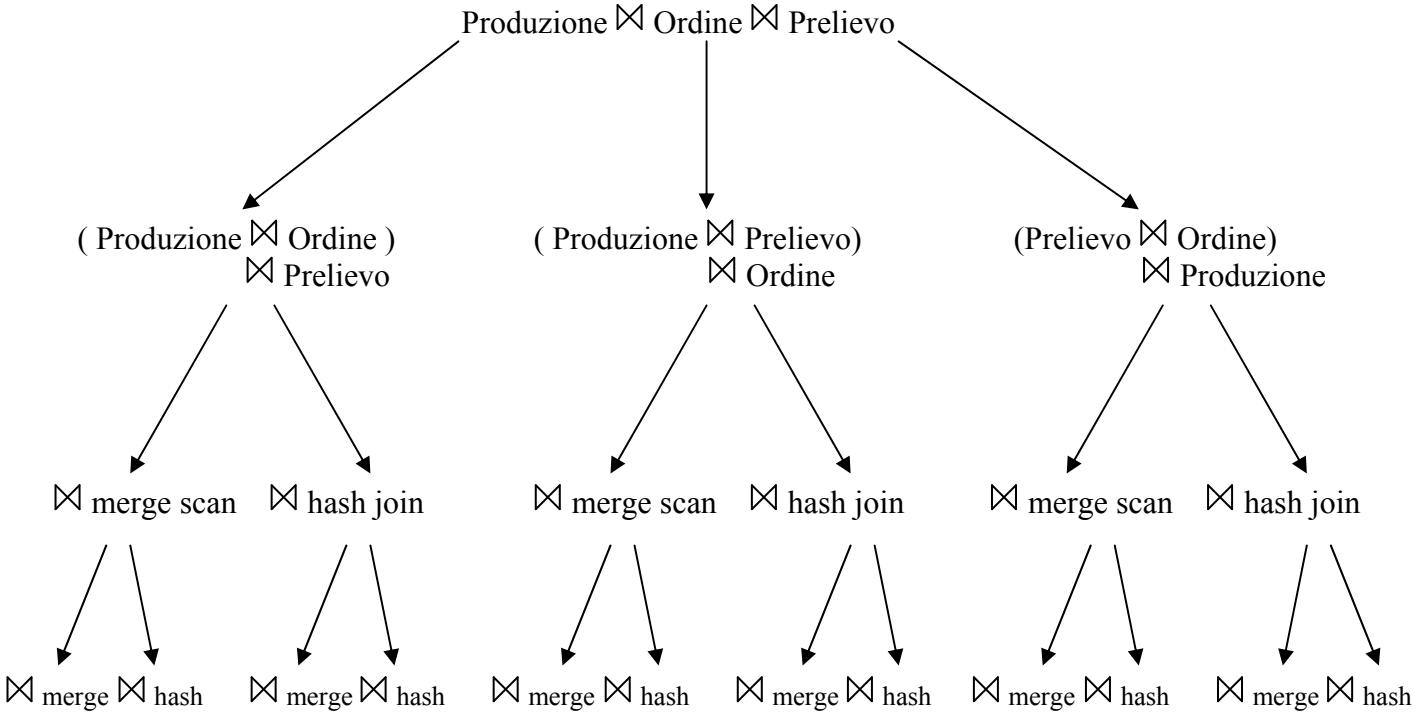
$SIZE(T_3) = 20$

Infine, poniamo $T_4 = \Pi_{Macchina}(T_2 \bowtie T_3)$

$CARD(T_4) = Min(CARD(T_2) * CARD(T_3) * (1 / VAL(NUMEROSERIE)), VAL(Macchina)) = 50$

$SIZE(T_4) = 10$

Albero delle decisioni:



La prima scelta riguarda l'ordine dei join, la seconda scelta indica il tipo del primo join scelto, l'ultima scelta indica il tipo dell'ultimo join scelto.

3) SQL:

```
select Cliente
from Ordine join Prelievo on
Ordine.Lotto=Prelievo.Lotto
join Commissione on
Ordine.Lotto=Commissione.Lotto
where Venditore='Bianchi' and NumeroSerie='3478'
```

Algebra relazionale:

$$\Pi_{\text{Cliente}} (\sigma_{\text{Venditore='Bianchi'} \wedge \text{NumeroSerie='3478'}} (\text{Ordine} \bowtie_{\text{Lotto=Or}} \rho_{\text{Or}\leftarrow\text{Lotto}} (\text{Prelievo}) \bowtie_{\text{Lotto=Or2}} \rho_{\text{Or2}\leftarrow\text{Lotto}} (\text{Commissione})))$$

Ottimizzazione algebrica:

$$\Pi_{\text{Client}} (\Pi_{\text{OrdNumber}} (\sigma_{\text{ProdNumber='3478'}} (\text{OrderDetail}) \bowtie_{\text{OrderNumber=Or1}} (\Pi_{\text{Or1}} (\sigma_{\text{Seller='White'}} (\rho_{\text{Or1}\leftarrow\text{OrderNumber}} (\text{Commission}))) \bowtie_{\text{OrderNumber=Or2}} (\Pi_{\text{Or2,Client}} (\rho_{\text{Or2}\leftarrow\text{OrderNumber}} (\text{Order})))$$

Poniamo $T_1 = \sigma_{\text{NumberoSerie='3478'}} (\text{Prelievo})$

$\text{CARD}(T_1) = (1 / \text{VAL}(\text{NumeroSerie})) * \text{CARD}(\text{Prelievo}) = (1/200.000) * 50.000 = 0,25$

$\text{SIZE}(T_1) = 15$

Poniamo $T_2 = (\Pi_{\text{Lotto}} (\sigma_{\text{Venditore='Bianchi'}} (\text{Commissione}))$

$\text{CARD}(T_2) = 200$

$\text{SIZE}(T_2) = 5$

Poniamo $T_3 = \Pi_{\text{Lotto}} (T_1 \bowtie T_2)$

$\text{CARD}(T_3) = 0,25 * 200 * (1/10.000) = 0,005$

$\text{SIZE}(T_3) = 5$

Poniamo $T_4 = \Pi_{\text{Lotto, Cliente}} (\text{Ordine})$

$\text{CARD}(T_4) = 10.000$

$\text{SIZE}(T_4) = 35$

Poniamo $T_5 = \Pi_{\text{Client}} (T_3 \bowtie T_4)$

$\text{CARD}(T_5) = 0,005 * 10.000 * (1/10.000) = 0,005$

$\text{SIZE}(T_5) = 30$

Notare che il valore di $\text{CARD}(T_i)$ può essere anche minore di 1: CARD è solo un valore statistico.

Esercizio 11.5

Elencare le condizioni (dimensioni delle tabelle, presenza di indici o di organizzazioni sequenziali o a hash) che rendono più o meno conveniente la realizzazione di join con i metodi nested loop, merge scan e hash-based; per alcune di queste condizioni, proporre delle formule di costo che tengano conto essenzialmente del numero di operazioni di ingresso/uscita come funzione dei costi medi delle operazioni di accesso coinvolte (scansioni, ordinamenti, accessi via indici).

Soluzione:

Dimensione delle tabelle: se siamo in una situazione in cui una delle tabelle è molto grande rispetto all'altra, può essere una buona soluzione l'utilizzo di un join con il metodo nested loop, usando la tabella maggiore come esterna e quella più piccola come interna.. Se le tabelle hanno la stessa dimensione il join con il metodo nested loop è conveniente in presenza di indici o hashing su una delle tabelle. Negli altri casi è meglio scegliere un merge-scan

Hashing: la presenza di una funzione di hash può suggerire l'utilizzo di un join hash-based o nested loop. La scelta dipende dalla dimensione di una tabella e dal numero di partizioni prodotte dalla funzione di hash.

Organizzazione sequenziale: naturalmente, se le due tabelle hanno un'organizzazione sequenziale sugli attributi del join, il merge-scan è la scelta migliore. Se solo una tabella ha un'organizzazione sequenziale il merge scan può essere ancora una buona scelta se non ci sono particolari condizioni (indici o hashing).

Indici:La presenza di indici in generale suggerisce un nested loop. Comunque, se la tabella con gli indici è molto piccola, o se l'indice è sparso, può essere meglio non utilizzare l'indice e fare uno scan completo.

Il costo di un join di tipo nested loop senza indici può essere espresso come:

C_{NL}=scan(T_1) + T_1 (scan (T_2))

Dove T_1 e T_2 sono il numero di tuple delle due tabelle, e scan(T) è il costo medio di un'operazione di scansione.

Se c'è un indice sulla tabella 2:

C_{NL}=scan(T_1) + T_1 (index (T_2))

Un merge scan di un'organizzazione non sequenziale ha un costo:

C_{MS}= order(T_1) + order(T_2) + scan(T_1) + scan(T_2)

Esercizio 11.6

```
select distinct C, L
from R1, R2, R3
where R1.C = R2.D and R2.F = R3.G and R1.B = R3.L and R3.H > 10
```

Mostrare un possibile piano di esecuzione (in termini di operatori di algebra relazionale e loro realizzazioni; prestate attenzione anche alla `DISTINCT`, in quanto le relazioni degli operatori non producono necessariamente insiemi, ma liste di tuple), giustificando brevemente le scelte più significative, con riferimento alle seguenti informazioni sulla base di dati:

- la relazione R1(ABC) ha 100.000 tuple, una struttura heap e un indice secondario su C;
- la relazione R2(DEF) ha 30.000 tuple, una struttura heap e un indice secondario sulla chiave D;
- la relazione R3(GHL) ha 10.000 tuple, una struttura heap e un indice secondario sulla chiave G.

Soluzione:

Un possibile piano di esecuzione per l'interrogazione in oggetto è basato essenzialmente sulla struttura fisica delle relazioni e sulla loro cardinalità.
La parte dell'interrogazione da analizzare è quindi quella della condizione, che viene riportata di seguito per comodità.

```
where R1.C = R2.D and R2.F = R3.G and R1.B = R3.L and R3.H > 10
```

Si effettuano i seguenti passaggi:

1. MERGE JOIN di tipo **scan** sulle relazioni R1 ed R2, in quanto R1.C e R2.D hanno un indice secondario.
2. MERGE JOIN di tipo nested loop sulle relazioni R2 e R3, ponendo R2.F = R3.G, mantenendo R2 esterno e R3 interno.
3. Selezione, tramite una scansione, degli attributi R1.B = R3.L e R3.H > 10

Esercizio 11.7

```
select distinct A, L
from T1, T2, T3
where T1.C = T2.D and T2.E = T3.F and T1.B = 3
```

Mostrare un possibile piano di esecuzione (in termini di operatori dell'algebra relazionale e loro realizzazioni), giustificando brevemente le scelte più significative, con riferimento alle seguenti informazioni sulla base dati:

- la relazione T1(ABC) ha 800.000 tuple e 100.000 valori diversi per l'attributo B, distribuiti uniformemente; ha una struttura heap e un indice secondario sulla chiave A
- la relazione T2(DE) ha 500.000 tuple; è definito un vincolo di riferimento fra l'attributo E e la chiave F della relazione T3: ha una struttura heap e un indice secondario sulla chiave D
- la relazione T3(FL) ha 1.000 tuple e ha una struttura hash sulla chiave F.

Soluzione:

Un possibile piano di esecuzione per l'interrogazione in oggetto è basato essenzialmente sulla struttura fisica delle relazioni e sulla loro cardinalità.
La parte dell'interrogazione da analizzare è quindi quella della condizione, che viene riportata di seguito per comodità

```
where T1.C = T2.D and T2.E = T3.F and T1.B = 3
```

Si effettuano i seguenti passaggi

1. Selezione, tramite una scansione, dell'attributo `T1.B = 3`. Questa soluzione ha un costo molto elevato, 800.000 accessi in memoria, ma permette di estrarre un numero molto basso di record, in quanto possiede 100.000 valori diversi e mediamente ci saranno 8 record per ogni occorrenza.
2. JOIN di tipo nested loop sulle relazioni T1 e T2, ponendo T1.C = T2.D, tenendo il risultato del passo precedente come tabella interna e T2 come esterna.
3. JOIN di tipo nested loop usando l'indice hash su T3.F

Esercizio 11.8 Calcolare il fattore di blocco e il numero di blocchi occupati da una relazione con $T = 1000000)$ di tuple di lunghezza fissa pari a $L = 200$ byte in un sistema con blocchi di dimensione pari a $B = 2$ kilobyte.

Soluzione

Dimensioni tabella (D_T):
$D_T = T * L$
$D_T = 1000000 * 200 = 200000000$ byte $= 190.73$ MB

Numero blocchi:
$N_B = D_T/B$
$N_B = 200000000/2048 = 97.66$ blocchi

Fattore di blocco:
$F_B = B/L$
$F_B = 2048/200 = 10.24$ record / blocco

Esercizio 11.9 Si considerino:

- un sistema con blocchi di $B = 1000$ byte e indirizzi ai blocchi di $p = 2$ byte; il sistema (in effetti un po' obsoleto) prevede indici, primari o secondari, a un solo livello e solo sul campo chiave;
- una relazione con $N = 1000000$ tuple, ciascuna di $l = 100$ byte, di cui $k = 8$ byte per la chiave.

Calcolare:

1. il numero dei blocchi necessari per un indice primario sul campo chiave;
2. il numero dei blocchi necessari per un indice secondario;
3. il numero di accessi a memoria secondaria necessari sequenziale sulla chiave;
4. il numero di accessi a memoria secondaria necessari utilizzi un indice primario;
5. il numero di accessi a memoria secondaria necessari utilizzi un indice secondario.

Calcolare il numero di accessi sia nel caso peggiore sia medio, nell'ipotesi che le ricerche abbiano successo in casi (cioè otto volte su dieci il record cercato esiste).

Soluzione

1. Dimensione record indice primario:
   | valore chiave | valore ind memoria | $KP = K + P$
   numero di blocchi (e non di record poiché la struttura è ordinata) a cui deve puntare l'indice primario (sparso):

   $NB = (K * N)/B = 8 * 1000000/1000 = 8000$ blocchi
   dimensione indice = $(K + P) * NB = (8 + 2) * 8000 = 80000$byte
2. Dimensione record indice secondario
   | valore chiave | valore ind memoria | $KP = K + P$
   numero di record (e non di blocchi perch la struttura non è ordinata) a cui deve puntare l'indice secondario (denso) pari a l
   dimensione indice $= l * (p + k) = 1000000 * 10 = 10000000$ byte
3. Supponendo l'assenza di buffer, il numero di accessi alla memoria secondaria necessari per un FULLSCAN della tabella sono pari, nel caso peggiore, al numero dei blocchi della tabella stessa e quindi:
   Accessi =$l * n/b = 100 * 1000000/1000 = 100000$
4. Utilizzando l'indice primario (separato dal file) si fa riferimento ad una struttura ordinata ad un solo livello
   numero blocchi indice = dimensione_indice$/B = 80000/1000 = 80blocchi$
   80 accessi nel caso peggiore.
   É necessario, infine, un ulteriore accesso per recuperare il record puntato.
5. Utilizzando l'indice secondario ad un solo livello la complessità è data dal numero di blocchi dell'indice. Si tiene inoltre conto dell'opportuna probabilità di esistenza del record.
   numero blocchi indice = dimensione_indice $/B = 10000000/1000 = 10000$blocchi

Essendo la struttura ordinata, la probabilità di visitare tutto l'indice è data, nel caso peggiore, dalla probabilità di presenza della chiave in ultima posizione (s).
Si ha accessi_medi $= 10000 * s = 8000$accessi
E' necessario, infine, un ulteriore accesso per recuperare il record puntato.

Esercizio 11.10 Si considerino un sistema con blocchi di dimensione $B = 1000$ byte e puntatori ai blocchi di $P = 2$byte e una relazione R(A,B,C,D,E) di cardinalità pari circa a $N = 1000000$, con tuple di $L = 50$byte e campo chiave A di $K = 5$ byte. Valutare i pro e i contro (in termini di numero di accessi a memoria secondaria e trascurando le problematiche relative alla concorrenza) relativamente alla presenza di un indice secondario sulla chiave A e di un altro, pure secondario, su B, in presenza del seguente carico applicativo:

1. inserimento di una nuova tupla (con verifica del soddisfacimento del vincolo di chiave), con frequenza $f_1 = 500$ volte al minuto;
2. ricerca di una tupla sulla base del valore della chiave A con frequenza $f_2 = 500$ volte al minuto;
3. ricerca di tuple sulla base del valore di B con frequenza $f_3 = 100$.

Soluzione
Si suppone la disponibilità di buffer che permettano di mantenere stabilmente in memoria due livelli per ciascun indice e considerando che la relazione possa essere memorizzata in forma contigua (assumendo un rapporto 100:1 fra tempo di posizionamento della testina e tempo di lettura).

Si suppone inoltre $L_B = 3$.
È necessario calcolare il costo delle quattro operazioni, in presenza e assenza degli indici. Notazioni:

N_T numero di blocchi della relazione T; è pari circa a $N/(B/L)$ = 50.000
$cont$ riduzione di costo dovuta alla contiguità; è pari a 100.
seq costo della scansione sequenziale $1 + (NT1)/cont$; è pari a circa 500.
$prof_A$ il fattore di blocco dell' indice è 1.000/7, circa 140 e quindi, con un riempimento di circa il 60-70%, il fan-out è circa 100 e quindi i livelli necessari sono 3.
$prof_B$ il fattore di blocco dellindice è 1.000/5, circa 200 e quindi, con un riempimento di circa il 60-70%, il fan-out è circa 135 e quindi i livelli necessari sono ancora 3.

Op_1 È influenzata da entrambi gli indici anche se in modo diverso. Vediamo i diversi comportamenti.

Nessun indice: è necessaria la scansione sequenziale per verificare il vincolo di chiave. Costo pari a $seq = 500$; nessun costo per la manutenzione degli indici.

Indice su A: ed accesso tramite lindice. Costo pari alla profondità dellindice su A, più uno (per accedere al blocco del file) meno due (grazie ai buffer); costo pari a 1.

Indice su B: scansione sequenziale per la verifica della chiave e accesso allindice per laggiornamento.
Indici sia su A sia su B: accesso ai due indici.

Op_2 È influenzata dal solo indice su A accessi diretti o sequenziali.

Op_3 È influenzata dal solo indice su B. E' sufficiente aggiungere al risultato precedente il fattore relativo alla molteplicità.

Esercizio 11.11

Alcuni DBMS permettono una tecnica di memorizzazione chiamata “co-clustering” o “clustering eterogeneo” in cui un file contiene record di due o più relazioni e tali record sono raggruppati (eventualmente, ma non necessariamente ordinati) secondo i valori di opportuni campi dell’una e dell’altra relazione. Ad esempio, date due relazioni

- Ordini(CodiceOrdine, Cliente, Data, Importo)
- LineeOrdine(CodiceOrdine, Linea, prodotto, Quantità, Importo)

Questa tecnica (con riferimento agli attributi CodiceOrdine delle due relazioni) permetterebbe una memorizzazione contigua di ciascun ordine con le rispettive “linee d’ordine”, cioè dei prodotti ordinati (ciascun ordine fa riferimento a più prodotti, ognuno su una “linea”).
Con riferimento all’esempio, indicare quali delle seguenti operazioni possono trarre vantaggio dall’uso di questa opportunità e quali ne possono essere penalizzate (spiegare la risposta anche in termini quantitativi, individuando valori opportuni per i principali parametri di interesse; supporre che siano utilizzati indici su CodiceOrdine, in tutti i casi, due per la memorizzazione tradizionale e uno nel caso di utilizzo del cluster eterogeneo):

1. stampa dei dettagli (cioè delle linee d’ordine) di tutti gli ordini (ordinati per codice)
2. stampa dei dettagli di un ordine
3. stampa delle informazioni sintetiche (codice, cliente, data, totale) di tutti gli ordini.

Soluzione:

1. Per quanto riguarda la prima operazioni, non ci sono dubbi che una struttura co-cluster porti notevoli benefici, in quanto è sufficiente posizionarsi sul primo record della relazione e scorrerne tutto il contenuto. Quindi, la prima operazione è VANTAGGIOSA.
2. La seconda operazione è più delicata, in quanto se si ha a disposizione una struttura hash o ad albero sulla relazione ordini è vantaggiosa, mentre se si ha una struttura sequenziale o disordinata è svantaggiosa. Quindi, in questo caso DIPENDE DALLA STRUTTURA DELLA RELAZIONE ORDINI.
3. La terza operazione è sicuramente svantaggiosa, perché in ogni caso devo accedere a tutte le tuple del co-cluster nonostante abbia bisogno dei soli dati della relazione ordini. Quindi, la terza operazione è SVANTAGGIOSA.

Esercizio 11.12 Si consideri una relazione IMPIEGATO (Matricola, Cognome, Nome, Data- Nascita) con un numero di ennuple pari a Nabbastanza stabile nel tempo (pur con molti inserimenti ed eliminazioni) e una dimensione di ciascuna ennupla (a lunghezza fissa) pari a L byte, di cui K per la chiave Matricola e C per il campo Cognome.
Supporre di avere a disposizione un DBMS che permetta strutture fisiche disordinate (heap), ordinate (con indice primario sparso) e hash e che preveda la possibilità di definire indici secondari e operi su un sistema operativo che utilizza blocchi di dimensione B e con puntatori ai blocchi di P caratteri.
Indicare quale possa essere lorganizzazione fisica preferita nel caso in cui le operazioni principali siano le seguenti:

1. ricerca sul cognome (o una sua sottostringa iniziale, abbastanza selettiva, si supponga che mediamente una sottostringa identifichi $S = 10$ ennuple) con frequenza f_1;
2. ricerca sul numero di matricola, con frequenza f_2;
3. ricerca sulla base di un intervallo della data di nascita (poco selettivo, restituisce circa il 5% delle ennuple), con frequenza f_3 molto minore di f_1 e f_2;

assumendo $N = 10000000$ ennuple, $L = 100$ byte, $K = 5$ byte, $C = 20$ byte, $B = 1000$ byte, $P = 4$ byte, $f_1 = 100$ volte al minuto, $f_2 = 2000$ volte al minuto, $f_3 = 1$ volte al minuto. Individuare le alternative più sensate sulla base di ragionamenti qualitativi e poi valutarle quantitativamente, ignorando i benefici derivanti dai buffer e dalla contiguità di memorizzazione.

Soluzione

Dividiamo la risposta in tre parti:
Scelta qualitativa delle strutture fisiche più indicate:

Le strutture fisiche debbono privilegiare le operazioni 1 e 2 che sono più frequenti della 3 (che per giunta è poco selettiva).

1. Per l'operazione 1 la struttura hash su cognome non va bene, perché le ricerche non sono esatte (cioè sono spesso fatte con parte del cognome). Poiché nel testo non è chiarito se oltre ad essere abbastanza stabile la dimensione sono limitati anche gli inserimenti e le eliminazioni, non si può dire se sia preferibile una struttura ordinata su cognome (con un indice sparso) oppure una disordinata (con indice denso su cognome); nel prosieguo, supponiamo che il grado di dinamicità sia limitato (o che sia possibile avere tempi morti per la riorganizzazione) e quindi sia possibile una struttura ordinata (che darebbe grandi benefici per le ricerche su sottostringa). In ogni caso, nell'interesse dell'operazione 2, sarebbe utile un indice secondario sulla matricola.
2. La struttura che meglio privilegerebbe l'operazione 2 è quella hash su matricola: la struttura hash è la più effciente per l'accesso diretto puntuale (cioè con un valore di chiave completo), a patto che la dimensione della relazione

sia abbastanza stabile; in questo caso entrambe le condizioni sono soddisfatte. In questo contesto, per supportare adeguatamente anche l'operazione 1, si potrebbe utilizzare un indice secondario sul cognome.

Valutazione analitica:
Ignoriamo gli arrotondamenti. Sono necessari $K = 5$ byte per la chiave e $P = 4$ byte per i puntatori ai blocchi.
In tutti i casi abbiamo: fattore di blocco del file (numero di record per blocco): $F_f = B/L = 10$
fattore di blocco dell'indice: $F_i = B/(K + P) = 111$
Esaminiamo le due alternative sopra illustrate
Costo unitario delle tre operazioni:

1. accesso per cognome tramite l'indice secondario; l'indice ha N record (poiché denso) e quindi l'albero ha N/F_i foglie e profondità pari a $log_{F_i}(N/Fi)$; un'operazione ha costo pari alla profondità aumentata di S (l'indice è secondario, quindi gli accessi a cognomi consecutivi portano a blocchi diversi);
2. accesso hash sulla matricola: costo costante, circa 1.3 accessi a blocchi in media;
3. accesso sequenziale: costo pari al numero di blocchi del file: $(N * 1.5)/F_f$ (il fattore 1.5 è motivato dagli spazi liberi necessari per la struttura hash).

costo totale:
$f_2 * 1.3 + f_1 * (S + log_{F_i}(N/F_i)) + f_3 * (N * 1.5)/F_f$

con $f_1 = 100$, $f_2 = 2000$, $f_3 = 1$

$2000*1.3+100*(10+log_{111}(10000000/111))+1*(10000000*1.5)/10 = 78800$

Costo unitario delle tre operazioni:

1. accesso per cognome tramite l'indice primario; l'indice ha N/F_f record (poiché sparso) e quindi l'albero ha $(N/F_f)/F_i$ foglie e profondità pari a $log_{F_i}((N/F_f)/F_i)$; un'operazione ha costo pari alla profondità aumentata di 1 (in questo caso cognomi consecutivi portano a record adiacenti);
2. accesso per matricola tramite l'indice secondario; come sopra, ma con un solo blocco da accedere invece di S, visto che le matricole sono "esatte": un'operazione ha costo pari $1 + log_{F_i}(N/F_i)$
3. accesso sequenziale come sopra (ma senza il fattore 1.5)

Costo totale:

$$f_1 * (1 + log_{F_i}((N/F_f)/F_i)) + f_2 * (1 + log_{F_i}(N/F_i)) + f_3 * N/F_f$$

$$f_1 = 100, f_2 = 2000, f_3 = 1$$

$100*(1+log_{111}((10000000/10)/111))+(2000*(1+log_{111}(10000000/111))+10000000/111 = 6107$

Esercizio 11.13 Illustrare brevemente vantaggi e svantaggi (relativamente a vari tipi di operazioni) delle seguenti strutture fisiche, definite con riferimento ad uno stesso attributo A non chiave:

1. indice primario (sparso) multilivello statico;
2. indice secondario multilivello statico;
3. B+ tree secondario;
4. hash.

Soluzione

1. Indice primario (sparso) multilivello statico: va bene per ricerche puntuali e su intervallo, ma soffre in caso di aggiornamenti.
2. Indice secondario multilivello statico: va bene per ricerche puntuali e su intervallo (un po' meno del precedente, perché l'indice è più grande e soprattutto perché il file non è ordinato, il che è rilevante perchè il campo non è chiave e sono considerate ricerche su intervalli), ma soffre in caso di aggiornamenti.
3. B+ tree secondario: come i precedenti, un po' meno effciente ma senza svantaggi particolari in presenza di aggiornamenti.
4. Hash: molto effciente per accessi puntuali, ma non per intervalli; degenera se le dimensioni variano signifcativamente.

Esercizio 11.14 Indicare, in ciascuno dei quattro casi dell'esercizio 11.13(sia in forma simbolica sia in forma numerica), il costo di operazioni di ricerca basate su A ("trovare i record con un certo valore per l'attributo A"), con riferimento ad una relazione R contenente $L = 100000000$ ennuple di $R = 25$ byte ciascuna, di cui $a = 12$ per l'attributo A. Si considerino mediamente $m = 3$ record con lo stesso valore su A; supporre che i blocchi abbiano dimensione $B = 2$KB, approssimabile come $B = 2000$e che i puntatori ai blocchi abbiano lunghezza $p = 4$.

Soluzione

1. $\text{LIV}_1 = 4$, dove LIV_1 è la profondità dell'indice, pari a $log_F L/(B/R) = 3$, dove F è il fattore di blocco dell'indice, pari a $B/(a + p)$;
2. $\text{LIV}_2 + m = 7$, dove LIV_2 è la profondità dell'indice, pari a $log_F L = 4$; si noti che il termine m è necessario qui perchè il file è disordinato, mentre non è necessario nel caso precedente, perché i record sono tutti nello stesso blocco (salvo poche eccezioni).
3. $\text{LIV}_3 + m = 8$, dove LIV_3 è la profondità dell'indice, pari a $log_{F'} L = 5$ dove $F' = 2/3F$ (supponendo un riempimento medio del 66%).
4. poco più di 1.