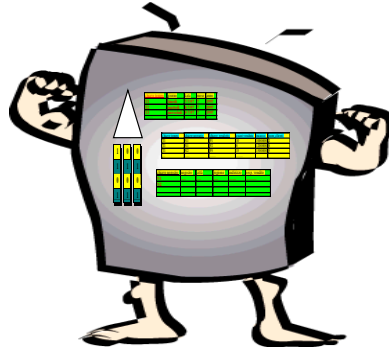


Organizzazioni dei dati - parte V



Dario Maio

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

Organizzazioni dei dati- parte V



1

Organizzazioni hash dinamiche

- Il limite principale delle organizzazioni hash statiche riguarda l'allocazione (statica) dell'area primaria. Nel caso di archivi (fortemente) dinamici un'allocazione statica è inadeguata, a causa o dell'eccessivo spreco di memoria (bassa utilizzazione) o del deterioramento delle prestazioni (alta utilizzazione). Inoltre, nel caso di overflow gestiti in area primaria, si ha il vincolo $d \leq 1$.
- Le organizzazioni hash dinamiche non presentano questi problemi, in quanto adattano l'allocazione dell'area primaria alla dimensione corrente dell'archivio.
- Esistono due grandi categorie di organizzazioni hash dinamiche:
 - + **con direttorio** (struttura ausiliaria), tra cui:
Virtual hashing, Dynamic Hashing, Extendible Hashing
 - + **senza direttorio**, tra cui:
Linear Hashing, Spiral Hashing

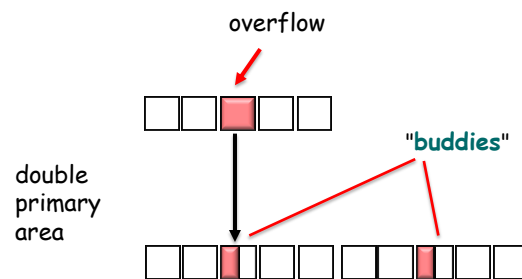
Organizzazioni dei dati- parte V



2

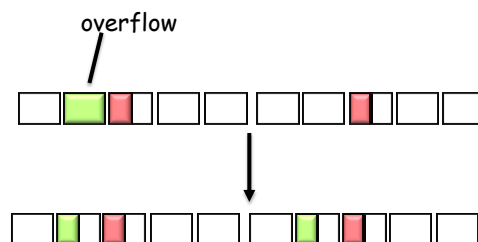
Virtual hashing (1)

- L'idea su cui si basa il Virtual hashing (Litwin 1978) è raddoppiare l'area primaria quando si verifica un overflow in un bucket, e ridistribuire i record tra il bucket saturo e il suo "buddy", facendo uso di una nuova funzione hash. In pratica si esegue lo "split" del bucket saturo.




Virtual hashing (2)

- Se, successivamente, qualche altro bucket nell'area primaria originale va in overflow, e il suo buddy non è ancora in uso, si ridistribuiscono i suoi record tra il bucket stesso e il buddy.



- Poiché, a un certo istante, solo alcuni buddy sono effettivamente in uso, è necessario fare ricorso a una struttura ausiliaria che permetta di determinare se occorre utilizzare la vecchia funzione hash o la nuova.



Virtual hashing : gestione area primaria (1)

Inizializzazione


- 1. Inizialmente vengono allocati NP_0 bucket di capacità C , e si fa uso di una funzione hash H_0 a valori in $[0, NP_0-1]$.
- 2. Si pone $L=0$; L rappresenta il numero di raddoppi eseguiti.
- 3. Si predispone un vettore binario V , di dimensione pari al numero di bucket, inizializzando tutti gli elementi al valore 1 ($V[j] = 1$ indica che il j -esimo bucket è in uso).
- 4. Dopo L raddoppi, l'area primaria contiene $NP = 2^L \times NP_0$ bucket, e il buddy di livello L del j -esimo bucket ($0 \leq j \leq 2^{L-1} \times NP_0 - 1$) è il bucket di indirizzo $j + 2^{L-1} \times NP_0$.

0	112 1176	1
1	512 3270	1
2	841 723	1
3	6849	1
4	7830 1075	1
5	6647 2840 2665	1
6	2385 286	1

$NP_0 = 7$
 $C = 3$


V

$$H_0 = k \bmod NP_0$$



Organizzazioni dei dati- parte V


5



Virtual hashing : gestione area primaria (2)

Split di un bucket

- 5. In caso di overflow del bucket j :
- 5.1 **Se** $L = 0$, **oppure** $L > 0$ ma il buddy di livello L è già in uso o non esiste ($L > 0$, ma $V[j + 2^{L-1} \times NP_0] = 1$ o $j \geq 2^{L-1} \times NP_0$)
 - 5.1.1 Si incrementa L , si raddoppia l'area primaria il vettore V ; i nuovi elementi di V valgono 0, eccetto $V[j + 2^{L-1} \times NP_0]$.
 - 5.1.2 Si introduce la nuova funzione hash H_L a valori nell'intervallo $[0, 2^L \times NP_0 - 1]$.
 - 5.1.3 Si ridistribuiscono le chiavi del bucket j facendo uso della funzione H_L .
- 5.2 **altrimenti**
 - 5.2.1 Si determina il buddy di livello minimo r ($1 \leq r \leq L$) non ancora in uso ($V[j + 2^{r-1} \times NP_0] = 0$).
 - 5.2.2 Si pone $V[j + 2^{r-1} \times NP_0] = 1$.
 - 5.2.3 Si ridistribuiscono le chiavi del bucket j facendo uso della funzione H_r .



Organizzazioni dei dati- parte V

6

Esempio: prima del raddoppio

$NP_0 = 7$
 $C = 3$

0	112
	1176
1	512
	3270
	841
2	723
3	6849
4	7830
	1075
	6647
5	2840
	2665
	2385
6	286

V

Si fa uso della famiglia di funzioni hash

$$H_L(k) = k \bmod (2^L \times NP_0)$$

Inserendo la chiave 3820 si ha:

$$H_L(k) = H_0(3820) = 5$$

quindi overflow poiché il bucket 5 è saturo

Si raddoppia l'area primaria e il vettore V e si ridistribuiscono le chiavi 3820, 2840, 2665, 2385 tra il bucket 5 e il suo buddy 12, facendo uso della funzione hash

$$H_1(k) = k \bmod 14$$

Organizzazioni dei dati- parte V

7

Esempio: dopo il raddoppio (1)

$NP_0 = 7$
 $C = 3$

0	112
	1176
1	512
	3270
	841
2	723
3	6849
4	7830
	1075
	6647
5	2840
	2665
	2385
6	286

V

Inserendo 3820

$NP_1 = 14$

0	112
	1176
1	512
	3270
	841
2	723
3	6849
4	7830
	1075
	6647
5	2665
	2385
6	286

7	
8	
9	
10	
11	
12	3820
	2840
13	

V

Organizzazioni dei dati- parte V

8

Esempio: dopo il raddoppio (2)

0	112	7	
	1176		
1	512	8	
	3270		
	841		
2	723	9	
3	6849	10	
4	7830	11	
	1075		
	6647		
5	2665	12	3820
	2385		2840
6	286	13	

$NP_1 = 14$

1
1
1
1
1
0
0
0
0
0
1
0

V

Se ora si deve inserire la chiave 3343 si ha $H_1(3343) = 11$. Poiché il bucket 11 non è ancora utilizzato si deve applicare $H_0(3343) = 4$.

Il bucket 4 è saturo e quindi si deve ancora ricorrere alla procedura di split, in questo caso **senza raddoppiare l'area primaria**, ma semplicemente ponendo $V[11]=1$ e di conseguenza ridistribuendo le chiavi.

Organizzazioni dei dati- parte V

9

Esempio: dopo il raddoppio (3)

0	112	7	
	1176		
1	512	8	
	3270		
	841		
2	723	9	
3	6849	10	
4	7830	11	3343
			1075
			6647
5	2665	12	3820
	2385		2840
6	286	13	

$NP_1 = 14$

1
1
1
1
1
0
0
0
0
0
1
1
0

V

$H_1(3343) = H_1(1075) = H_1(6647)$

$H_1(7830) = 4$.

dopo l'inserimento di 3343

Organizzazioni dei dati- parte V

10

Note sul raddoppio

in uso e saturo

non in uso

Inserimento chiave k

se $H_2(k)=2$ che è un bucket saturo il buddy di livello 1 non è in uso allora si pone in uso $V[9]=1$ e si ridistribuisce con H_1

se la ridistribuzione fallisce il prossimo buddy è quello di livello 2, si pone in uso $V[16]=1$ e si ridistribuisce con H_2

se fallisce ancora si raddoppia !

Organizzazioni dei dati- parte V

11

Virtual hashing: funzioni hash

- Il virtual hashing richiede una serie di funzioni hash $H_0, H_1, \dots, H_L, \dots$ che soddisfino le seguenti condizioni:
 - Range condition**
La funzione H_L deve essere a valori in $[0, 2^L \times NP_0 - 1]$.
 - Split condition**
Per ogni $L > 0$, per ogni valore di chiave k , e per ogni valore di $H_L(k)$, deve valere la relazione:

$$H_L(k) = H_{L-1}(k) \text{ oppure } H_L(k) = H_{L-1}(k) + 2^{L-1} \times NP_0$$
 ovvero lo split di un bucket deve lasciare una chiave nel bucket stesso, o allocarla nel buddy.
- La famiglia di funzioni $H_L(k) = k \bmod (2^L \times NP_0)$ evidentemente soddisfa la prima condizione; si dimostra che soddisfa anche la condizione di split.

Organizzazioni dei dati- parte V

12

Ricerca e inserimento di una chiave

0	112
	1176
1	512
	3270
	841
2	723
3	6849
4	7830
	1075
	6647
5	2665
	2385
6	286

7	
8	
9	
10	
11	
12	3820
	2840
13	

* Per cercare un valore di chiave è necessario conoscere con quale funzione hash è stato allocato. Il vettore V è sufficiente allo scopo
 * L'algoritmo di ricerca restituisce l'indirizzo del bucket in cui si può trovare il valore di chiave cercato (o che si vuole inserire).
 * La ricerca si attiva con **Address(L,k)**, considerando cioè il numero di raddoppi eseguiti.

```

function Address(r:integer;k:chiave):integer;
{
  if r<0 then la chiave non esiste
  else if V[Hr(k)] = 1
    then Address:=Hr(k)
    else Address:=Address(r-1,k)
}
          
```

$NP_1 = 14$

Organizzazioni dei dati- parte V

13

Dynamic hashing

- Il metodo dynamic hashing (Larson 1978) evita di ricorrere a tecniche di raddoppio (che riducono l'utilizzazione e sono causa di appesantimenti non trascurabili nel momento in cui l'area primaria viene raddoppiata), facendo uso di una struttura ausiliaria (**directory**) organizzata come un **trie binario**.
- L'idea di base, comune al metodo extendible hashing, consiste nell'adottare una funzione hash che, dato un valore di chiave k_i , restituisce non un indirizzo di bucket, bensì una stringa binaria, detta **pseudo-chiave**:

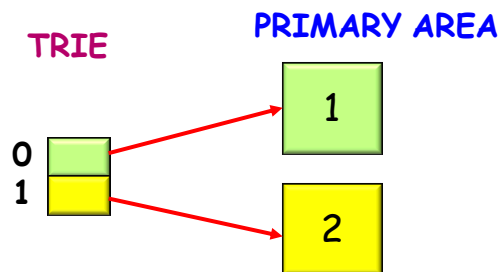
$H(k_i) = b_{i,0}, b_{i,1}, b_{i,2}, \dots$
- La situazione ideale è quella in cui l'insieme di pseudo-chiavi da gestire è tale per cui vi è una ripartizione bilanciata per ogni posizione considerata, cioè $Pr\{b_{i,j} = 1\} = 1/2$.
- Un semplice metodo per ottenere le pseudo-chiavi consiste nell'utilizzare k_i come **il seme di un generatore** di numeri binari pseudo-casuali.

Organizzazioni dei dati- parte V

14

Dynamic hashing: uso del trie

- Il trie serve a organizzare la ricerca, e i suoi nodi foglia indirizzano i bucket dell'area dati. Per cercare (o inserire) un valore di chiave si segue, fino a una foglia, il cammino del trie corrispondente alla pseudo-chiave.
- **Esempio:** il bucket 1 contiene tutti i valori di chiave la cui pseudo-chiave è del tipo 0..., e il bucket 2 quelli con pseudo-chiave 1...

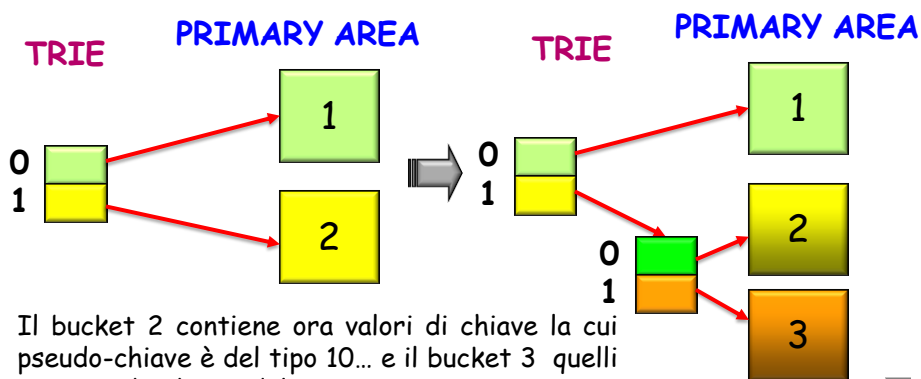


Organizzazioni dei dati- parte V

15

Dynamic hashing: espansione

- L'espansione dell'area primaria avviene aggiungendo un bucket alla volta, ridistribuendo i record tra il bucket saturo e il suo buddy, e aggiungendo un nodo al trie.
- **Esempio:** se si deve eseguire lo split del bucket 2 :



Organizzazioni dei dati- parte V

16

Dynamic hashing: prestazioni

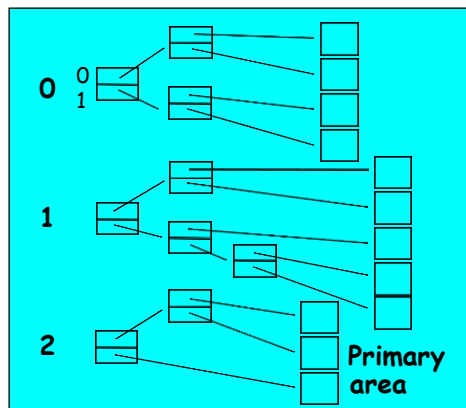
- Se il trie è in memoria centrale è sufficiente un singolo accesso per recuperare un record, altrimenti le prestazioni dipendono dal bilanciamento, in termini di numero di nodi indice da reperire.
- Le prestazioni nel caso peggiore non sono buone. Infatti, dipendentemente dall'insieme di pseudo-chiavi, l'inserimento di un nuovo record può comportare più di uno split.
- Se dopo una cancellazione in un bucket, il numero di record contenuti nel bucket e nel suo buddy diventa minore o uguale alla capacità C , i bucket vengono fusi, e si elimina un nodo foglia dal trie.
- L'utilizzazione media è di circa il 70%.

Dynamic hashing: variante

- Una variante, suggerita dallo stesso Larson, consiste nell'allocare inizialmente NP bucket e fare uso di una qualsiasi funzione hash statica H_0 . A seguito di overflow, si genereranno NP trie, le cui radici sono indirizzate da H_0 .

$$H_0(k) = k \bmod 3$$

NP = 3 tries



Extendible hashing

- Extendible hashing (Fagin, Nievergelt, et al. 1979) è un'organizzazione molto simile al dynamic hashing, da cui si differenzia per il modo con cui gestisce il direttorio.
- ✚ Garantisce il reperimento di un record con non più di due accessi alla memoria secondaria.

✚ Il direttorio è un insieme di 2^p celle che hanno indirizzi nell'intervallo $[0, 2^p - 1]$ e $p \geq 0$ è detta *profondità* del direttorio.

- ✚ Una funzione hash associa a ogni chiave una pseudo-chiave binaria,

$$H(k_i) = \dots, b_{i,2}, b_{i,1}, b_{i,0}$$

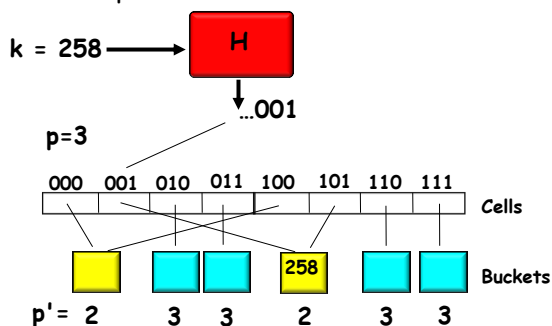
di cui si considerano i p bit meno significativi per accedere *direttamente* a una delle 2^p celle, ognuna contenente un puntatore a un bucket.



Extendible hashing: direttorio

- Ogni bucket ha una *profondità locale* $p' \leq p$ (valore mantenuto nel bucket) che indica il numero *effettivo* di bit usati per allocare le chiavi nel bucket stesso.

- ✚ **Esempio:** Il bucket che contiene la chiave 258 ha $p' = 2$. Pertanto contiene sia chiavi con pseudo-chiave del tipo $\dots 001$ sia chiavi con pseudo-chiave del tipo $\dots 101$.



Extendible hashing: split

- Inizialmente si ha un solo bucket, $p' = 0$ e $p = 0$.

Se si deve eseguire lo split di un bucket a profondità locale p' , si presentano 2 casi:

$p' < p$

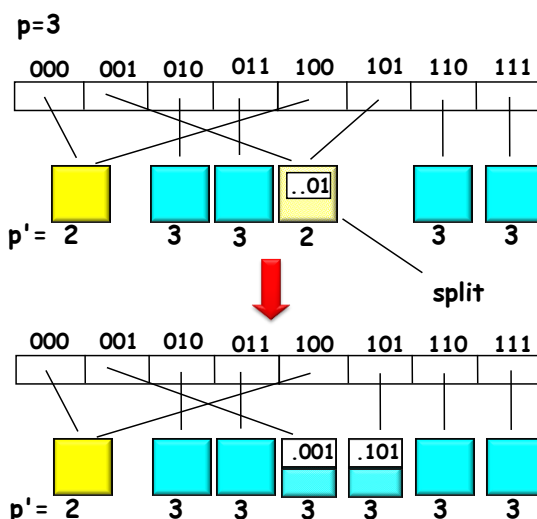
- ✦ si alloca un nuovo bucket e si distribuiscono le chiavi tra i due bucket facendo uso del $(p'+1)$ -esimo bit delle pseudo-chiavi. Per i due bucket si pone a $p'+1$ il valore della profondità locale.
- ✦ poiché $p' < p$, esiste almeno una cella che può indirizzare il nuovo bucket. Si modifica pertanto il puntatore della/e cella/e.

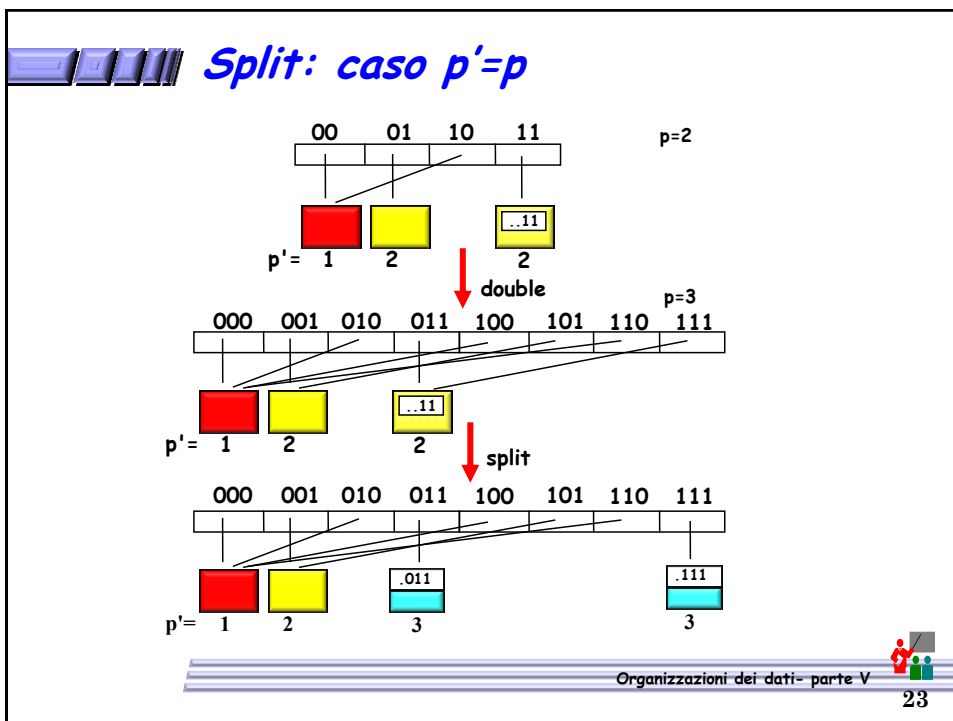
$p' = p$

- ✦ si raddoppia il direttorio, e si incrementa p di 1. Si copiano i valori dei puntatori nelle nuove celle corrispondenti
- ✦ si esegue lo split come nel caso $p' < p$.



Split: caso $p' < p$



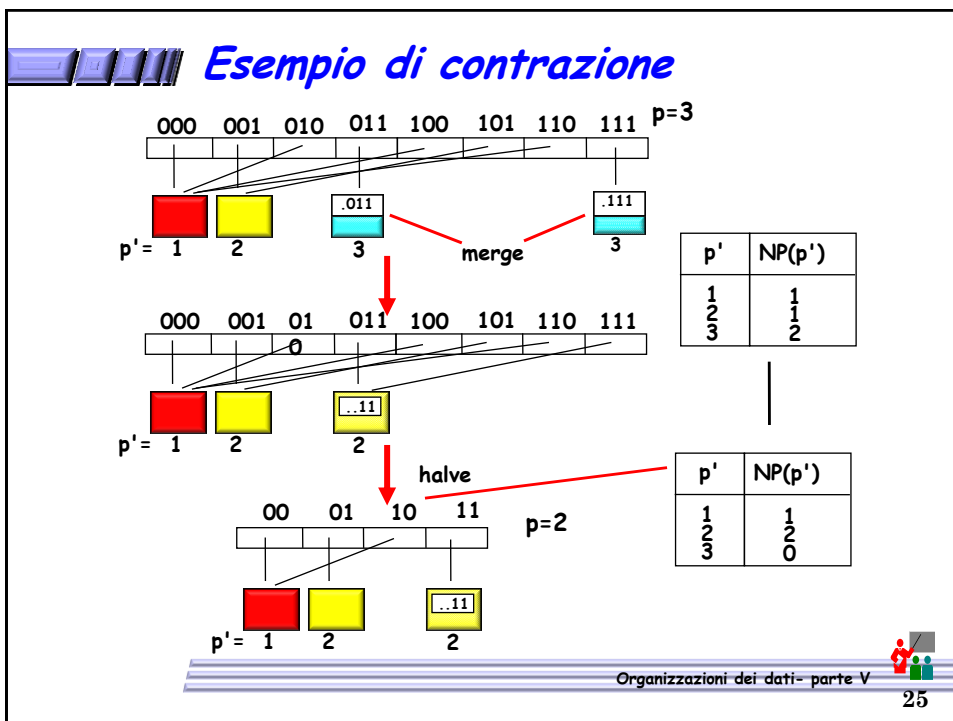


Contrazione del direttorio

- Se si cancella un record in un bucket a profondità p' , e il numero di record contenuti nel bucket e nel suo buddy diventa minore o uguale alla capacità C , i bucket vengono fusi. Per il bucket risultante la profondità locale vale $p'-1$.
- Se si fondono gli unici due bucket a profondità $p' = p$, è possibile contrarre il direttorio, dimezzandolo.
- Poiché verificare che non esistono più bucket a profondità p richiede, nel caso peggiore, di esaminare *metà* dei bucket, è conveniente fare uso di una **tabella delle profondità locali** che, per ogni valore $p' \leq p$, mantiene il numero, $NP(p')$, di bucket a profondità p' .

Organizzazioni dei dati- parte V

24



Linear hashing

- L'idea di base del Linear hashing (Litwin 1980), e delle altre organizzazioni hash dinamiche a "espansione lineare", è:
- Non si esegue lo split del bucket in cui si è verificato un overflow, ma si suddivide un altro bucket, scelto secondo un criterio prefissato.
- Le principali conseguenze sono:
 - ✦ Non è necessario un direttorio, in quanto si è a conoscenza dei bucket che sono stati suddivisi.
 - ✦ Occorre gestire l'overflow (in area primaria o separata).
 - ✦ L'area primaria cresce "linearmente" (non si hanno raddoppi).
- Gestendo gli overflow in area separata, ed eseguendo gli split in sequenza ordinata (a partire dal bucket 0), si ha lo schema base del linear hashing.

Organizzazioni dei dati- parte V 26

Gestione area primaria (1)

- ✦ Inizialmente si allocano NP_0 bucket e si usa la funzione hash

$$H_0(k) = k \bmod NP_0$$
- ✦ Si mantiene un puntatore (detto **split pointer**, SP) al prossimo bucket che deve essere suddiviso. Inizialmente $SP = 0$.
- ✦ Se si verifica un overflow si aggiunge in coda un bucket di indirizzo $NP_0 + SP$, si riallocano i record del bucket SP (inclusi quelli eventualmente presenti in area di overflow) facendo uso della nuova funzione hash:

$$H_1(k) = k \bmod (2 \times NP_0)$$
 e si incrementa $SP (= SP + 1)$.
- ✦ Dopo NP_0 overflow si è operata un'**espansione completa** dell'area primaria, in quanto il numero di bucket è ora pari a $2 \times NP_0$.

continua...

Organizzazioni dei dati- parte V

27

Gestione area primaria (2)

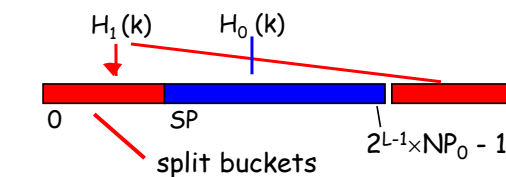
- Dopo un'espansione completa, ci si predispone per una nuova espansione ponendo
 $SP=0 \quad H_0(k) = H_1(k) \quad H_1(k) = k \bmod (2^2 \times NP_0)$.
- Durante la L-esima espansione si fa uso delle funzioni hash

$$H_0(k) = k \bmod (2^{L-1} \times NP_0)$$

$$H_1(k) = k \bmod (2^L \times NP_0)$$

e l'indirizzo dell'home bucket di una chiave si calcola come:

if $H_0(k) \geq SP$
then Address := $H_1(k)$ else Address := $H_0(k)$



Organizzazioni dei dati- parte V

28

Esempio

0	112	
	1176	
1		
2	841	
	30	
	289	
3	6849	
	72	
	717	
	731	
4	7830	
	1075	
	6647	
5	2840	
	2665	
	2385	
6	286	
7	287	
	147	
8	848	
	512	
	3270	

$NP_0=7, C=3, C_{ov}=2$

Gli overflow nei bucket 2 e 3 hanno in precedenza causato lo split dei bucket 0 e 1.

L'inserimento della chiave 3820 genera:

- ✚ overflow nel bucket 5,
- ✚ split del bucket 2,
- ✚ allocazione del bucket 9.

0	112	
	1176	
1		
2	841	
	30	
	72	
3	6849	
	717	
	731	
4	7830	
	1075	
	6647	
5	2840	
	2665	
	2385	
6	286	
7	287	
	147	
8	848	
	512	
	3270	
9	569	
	289	

Organizzazioni dei dati- parte V
29

Linear hashing: pregi e difetti

■ **Pregi**

- ✚ L'assenza di un direttorio e la politica di gestione degli split rendono semplice la realizzazione della struttura.
- ✚ La gestione dell'area primaria (espansione e contrazione) è immediata, in quanto i bucket vengono sempre aggiunti (e rimossi) in coda.

■ **Difetti**

- ✚ L'utilizzazione della memoria allocata, data da:

$$u = \frac{NR}{C \times NP + C_{ov} \times NP_{ov}}$$

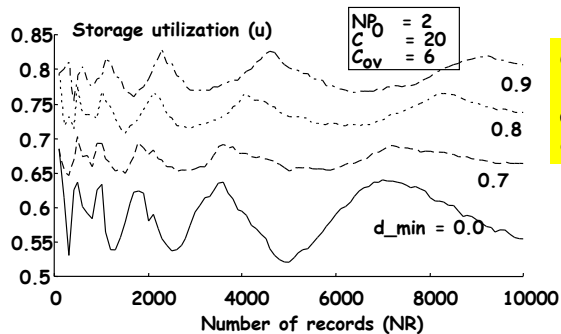
è decisamente bassa (variabile tra 0.5 e 0.7);

- ✚ La gestione dell'area di overflow presenta problemi simili a quelli di un'area primaria statica.
- ✚ Le catene di overflow relative ai bucket di indirizzo maggiore, non ancora suddivisi, possono diventare molto lunghe.

Organizzazioni dei dati- parte V
30

Linear hashing: utilizzazione

- Rispetto alla versione di base, che esegue lo split ogni qualvolta si verifica un overflow, sono possibili le seguenti varianti:
- Controllo del carico:** dopo la fase iniziale di caricamento negli NP_0 bucket, gli split avvengono ogni L inserimenti.
- Split controllato:** si esegue lo split solo quando il fattore di caricamento, d , raggiunge un valore di soglia d_{min} (es. $d_{min} = 0.8$).



Ciclicità del linear hashing: i casi peggiori si verificano a $1/3 \div 2/3$ di un'espansione completa

Organizzazioni dei dati- parte V



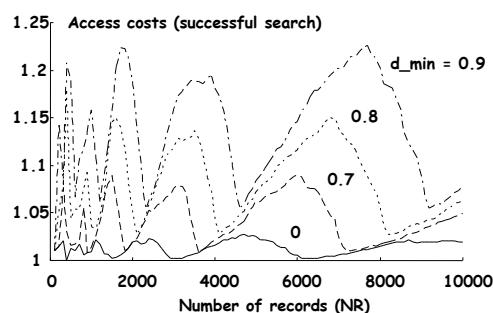
31

Costi di accesso

- All'aumentare dell'utilizzazione di memoria aumentano anche i costi di ricerca, in quanto tende ad aumentare il numero di record in overflow. Risultati qualitativamente simili si ottengono anche nel caso di ricerca con insuccesso.
- Anche per i costi di accesso si può osservare come le prestazioni dipendano dal valore specifico di NR e, quindi, di NP .

Si noti tuttavia che, a parità di d_{min} , i costi di accesso diminuiscono all'aumentare dell'utilizzazione di memoria, e viceversa, come facilmente rilevabile dal confronto dei due grafici relativi.

Esempio: per $NR \approx 5000$ e $d_{min} = 0$ si ha un minimo relativo dell'utilizzazione e un massimo relativo dei costi di accesso.



Organizzazioni dei dati- parte V



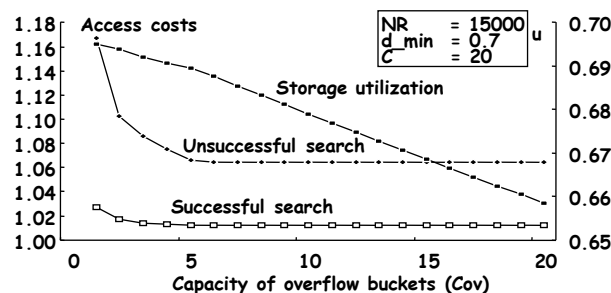
32

Capacità bucket di overflow

- Litwin fornisce, come indicazione pratica per la scelta della capacità dei bucket di overflow un valore pari a

$$C_{ov} \cong 1/5 \div 1/3 C$$

- Aumentare C_{ov} riduce la lunghezza (in bucket) delle catene di overflow, ma questa lunghezza non può comunque essere minore di 1 (se vi sono overflow). Pertanto oltre un certo valore si ha solo un inutile spreco di memoria.



Organizzazioni dei dati- parte V



33

Varianti del linear hashing

- **Linear hashing con espansioni parziali** (Larson 1980, Ramamohanarao & Lloyd 1982).

✚ la tecnica parte dalla considerazione che un bucket già suddiviso contiene, in media, un numero di record pari alla metà di quello contenuto in un bucket ancora da dividere. Ciò porta a una bassa utilizzazione della memoria e a una distribuzione degli overflow non uniforme. L'idea di base delle tecniche di espansione parziale è anticipare gli split, dividendo un "gruppo" di più bucket alla volta.

- **Linear hashing ricorsivo** (Ramamohanarao & Sacks-Davis 1984)

✚ La caratteristica principale riguarda la gestione dell'area di overflow, che è organizzata dinamicamente facendo uso del Linear hashing stesso. Si creano vari livelli di file hash dinamici (in media non più di 3), con il file al livello h ($h = 0, \dots, L$; $h = 0$ area primaria) che memorizza i suoi overflow nel file a livello $h + 1$. Al livello h si mantiene lo split pointer SP_h .

Organizzazioni dei dati- parte V



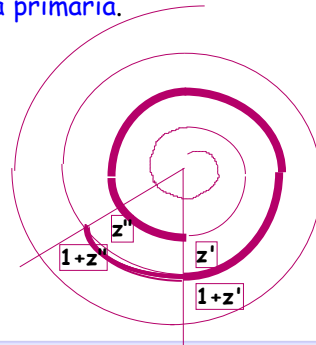
34

Spiral hashing

- Con il linear hashing c'è una maggiore probabilità di avere overflow dai blocchi non ancora suddivisi durante l'espansione corrente; infatti, l'uso di una funzione hash uniforme ha come conseguenza che ogni valore di $H_0(k)$ è ugualmente probabile, ma i bucket per cui si ha $H_0(k) < SP$ sono già stati suddivisi. La tecnica spiral hashing (Martin '79) cerca di risolvere questo problema impiegando una **funzione di tipo esponenziale, che consente di memorizzare i record più densamente nell'estremo iniziale dell'area primaria**.

L'organizzazione deve il suo nome (e quello di alcuni suoi parametri) al fatto che lo spazio di memoria viene pensato come una spirale, invece che come una retta, e l'area primaria come una rivoluzione della **spirale**, univocamente definita da un **angolo z** .

Esempio di Area Primaria



Organizzazioni dei dati- parte V



35