



Organizzazioni dei dati - parte IV




Dario Maio

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



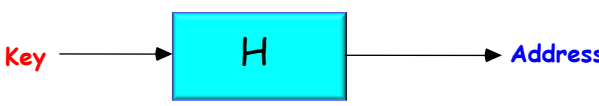
Organizzazioni dei dati- parte IV

1



Organizzazioni Hash


- A differenza delle tecniche di tipo "tabellare", in cui l'associazione (chiave, indirizzo) è mantenuta in forma esplicita, un'organizzazione hash utilizza una **funzione hash**, H , che trasforma ogni valore di chiave in un indirizzo.



```

graph LR
    Key[Key] --> H[H]
    H --> Address[Address]
  
```

- Salvo casi particolari, le funzioni hash non sono iniettive, cioè non rispettano la proprietà $k_1 \neq k_2 \Rightarrow H(k_1) \neq H(k_2)$, e quindi possono verificarsi collisioni:
 - k_1 e k_2 collidono $\Leftrightarrow H(k_1) = H(k_2)$
- Ogni indirizzo generato dalla funzione hash individua una **pagina logica**, o **bucket**. Il numero di "elementi" (valori di chiave se l'organizzazione è un indice, record dati se l'organizzazione è primaria) che possono essere allocati nello stesso bucket determina la **capacità**, C , dei bucket.



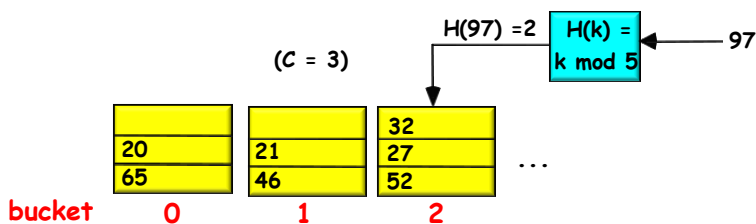
Organizzazioni dei dati- parte IV

2



Collisioni (bucket overflow)

- ✚ Se una chiave viene assegnata a un bucket che già contiene C chiavi, si verifica un **overflow** (trabocco).
- ✚ La presenza di overflow può richiedere, dipendentemente dalla specifica organizzazione, l'uso di un'area di memoria separata, detta appunto **area di overflow**.
- ✚ L'area di memoria costituita dai bucket indirizzabili dalla funzione hash è detta **area primaria**.



Organizzazioni hash statiche e dinamiche (1)

- Una funzione hash deve essere **suriettiva**, e quindi generare NP indirizzi $(0, 1, \dots, NP-1)$, tanti quanti sono i bucket dell'area primaria.
- ✚ Se il valore di NP è, per una data organizzazione, costante, l'organizzazione è detta **statica**. In questo caso, il dimensionamento dell'area primaria è parte integrante del progetto dell'organizzazione.
- ✚ Viceversa, se l'area primaria può espandersi e contrarsi, per adattarsi meglio al volume effettivo dei dati da gestire, allora l'organizzazione è detta **dinamica**. In questo caso si rendono necessarie più funzioni hash.
- Le organizzazioni statiche sono state sviluppate a partire dagli anni 50 sia per memoria centrale sia per memorie ausiliarie. Le prime organizzazioni hash dinamiche risalgono alla fine degli anni 70.

Organizzazioni hash statiche e dinamiche (2)

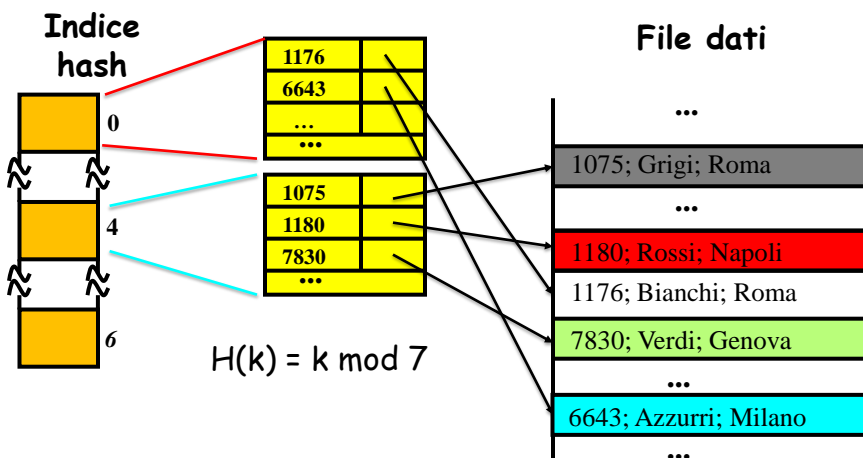
- Per entrambi i tipi di organizzazione, vi sono aspetti comuni che meritano considerazione:
 - ✦ Scelta della funzione hash H
 - ✦ Politica di gestione degli overflow
 - ✦ Capacità C dei bucket dell'area primaria
 - ✦ Capacità C_{ov} dei bucket dell'eventuale area di overflow (non necessariamente uguale a C)
 - ✦ Utilizzazione della memoria allocata
- Per quanto riguarda la distinzione tra organizzazioni primarie e secondarie, quelle hash sono usualmente del primo tipo.
- L'uso di indici hash è fortemente sconsigliato nei casi in cui sono possibili interrogazioni di intervallo, in quanto le funzioni hash in generale **non preservano l'ordine**, ovvero non sono funzioni monotone.

Organizzazioni dei dati- parte IV

5

Esempio indice hash

- Funzione hash che non preserva l'ordinamento



Organizzazioni dei dati- parte IV

6

Organizzazioni hash statiche (1)

- Un semplice esempio in cui:
 - ✚ le chiavi sono numeri naturali.
 - ✚ l'area primaria consiste di $NP = 5$ bucket di capacità $C = 5$.
 - ✚ la funzione hash è: $H(k_i) = k_i \bmod 5$
 - ✚ gli overflow sono gestiti allocando, per ogni bucket dell'area primaria, uno o più **bucket di overflow**, di capacità $C_{ov} = 5$, collegati a lista.

0					
1	21	16	31		
2	32	77	17	12	2
3	53				
4	69	24			

22					
----	--	--	--	--	--

Primary area
Overflow area

Organizzazioni dei dati- parte IV

7

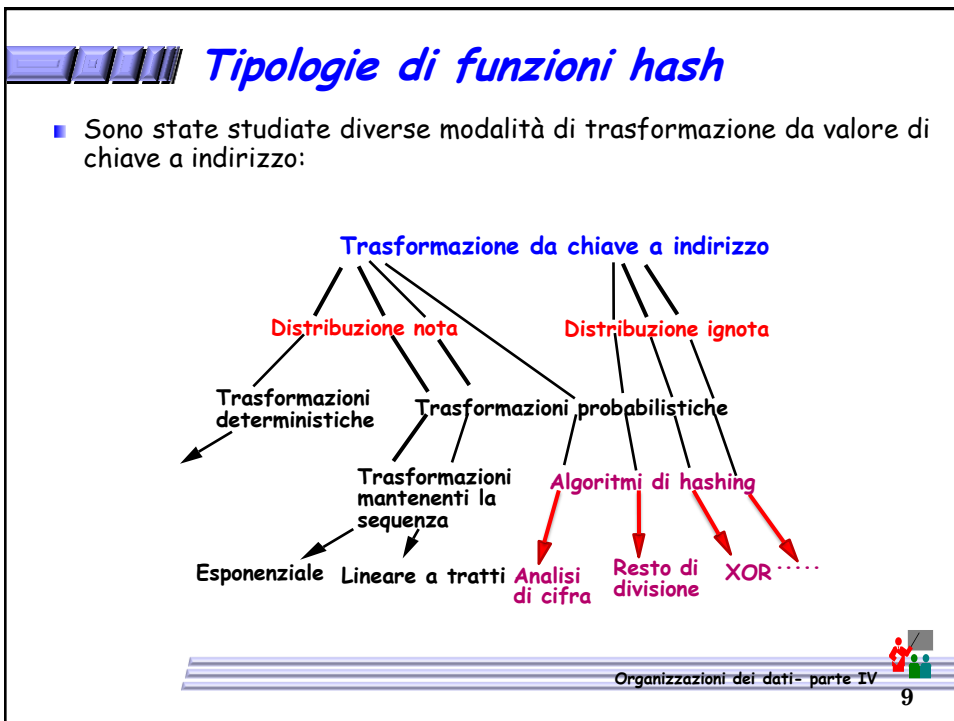
Organizzazioni hash statiche (2)

- Un file hash che memorizza NR record in bucket di capacità $C = C_{ov}$, e la cui area primaria consiste di NP bucket, comporta, nell'ipotesi di perfetta ripartizione dei record sullo spazio degli NP indirizzi, che:
 - ✚ ogni indirizzo è generato NR / NP volte
 - ✚ ogni catena consiste di $NR / (NP \times C)$ bucket
 - ✚ Il costo di ricerca di un record risulta pertanto proporzionale a $NR / (NP \times C)$
- **Esempio:**

con $NR = 10^6$, $C = 10$, $NP = 25000$, una ricerca con successo accede in media a 2 bucket. Questo è anche il numero di operazioni di I/O da eseguire, a condizione che un bucket sia reperibile con una singola lettura da disco.

Organizzazioni dei dati- parte IV

8



9

Funzioni hash perfette

- In molti casi è desiderabile utilizzare funzioni hash perfette (PHF) le quali mappino un insieme statico di chiavi in bucket (di capacità unitaria) senza dar luogo a collisioni.
 - Esempi d'uso:** archivi statici, parole riservate di un linguaggio di programmazione, comandi di sistema operativo, help on-line, ecc.
- Le funzioni hash perfette sono "rare". Ad esempio, date le $NK=31$ chiavi corrispondenti alle più comuni parole inglesi, e $NP = 41$ bucket di capacità unitaria, meno di una funzione su 10^7 è perfetta (iniettiva), in quanto:


$$\binom{41}{31} = \text{n. modi di scegliere 31 bucket}$$

$$31! = \text{n. modi di assegnare le 31 chiavi ai 31 bucket}$$

$$41^{31} = \text{n. funzioni da } [1..31] \text{ a } [1..41]$$

$$\frac{\binom{41}{31} \times 31!}{41^{31}} \approx 0.93 \times 10^{-7}$$

- Se la tabella hash ha dimensione minima, pari al numero delle chiavi, si parla più propriamente di **funzioni hash perfette minimali (MPHF)**.
- Se, infine, viene mantenuto l'ordinamento delle chiavi, si parla di **Order Preserving MPHF (OPMPHF)**.

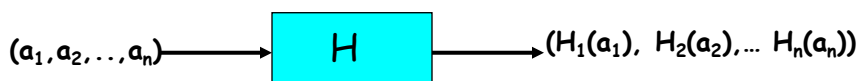
Organizzazioni dei dati- parte IV 

10

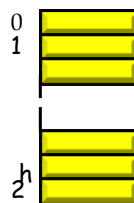


Hash per chiavi multiple (1)

- Nel seguito per semplicità si farà sempre riferimento a hashing su singolo attributo; è tuttavia possibile costruire **strutture hash per più attributi**.
- Dato un insieme di record con **n** chiavi di ricerca A_1, A_2, \dots, A_n e un'area primaria di 2^h bucket (indirizzabile con **h** bit), si suddivide **h** in **n** gruppi uno per ogni attributo, dedicando **b_i** bit al generico gruppo i-mo.



$$0 \leq H_i(a_i) \leq 2^{b_i} - 1$$



Organizzazioni dei dati- parte IV



11



Hash per chiavi multiple (2)

■ Esempio:

$A_1: \text{int}(5), A_2: \text{int}(9), A_3: \text{string}(10)$

area primaria di $2^9=512$ bucket (indirizzabile con **h=9** bit)

b₁=4 per A_1 , **b₂=3** per A_2 , **b₃=2** per A_3

$H_1 = A_1 \bmod 16$ $H_2 = A_2 \bmod 8$

$H_3 = (\text{num. caratteri} \# \text{ blank di } A_3) \bmod 4$

$(58651, 130326734, \text{"mamma"}) \rightarrow 11 \mid 6 \mid 1$ in binario: **1011** | **110** | **01**
bucket #377

- ⚡ Nelle interrogazioni parzialmente specificate, per ogni attributo specificato si riduce la ricerca di un fattore 2^{b_i}

Es. se mancano condizioni su A_3

$(58651, 130326734, ?) \rightarrow 1011 \mid 110$

00
01
10
11

si devono visitare
4 bucket

Organizzazioni dei dati- parte IV



12



Distribuzioni note a priori

- Esistono casi in cui la distribuzione dei valori di chiave è nota a priori. In questo caso è possibile fare uso di funzioni hash ad hoc.

Esempio: si abbiano NR record, da allocare in NP bucket, i cui valori di chiave sono numeri naturali distribuiti uniformemente nell'intervallo $[K_{min}, K_{max}]$ ($NR \leq K_{max} - K_{min} + 1$).

- L'idea è suddividere $[K_{min}, K_{max}]$ in NP sottointervalli. Poiché la distribuzione è uniforme, i sottointervalli avranno tutti la stessa ampiezza.
- La funzione hash avrà quindi la forma:

$$H(k_i) = \left\lfloor \frac{k_i - K_{min} + 1}{K_{max} - K_{min} + 1} \times NP \right\rfloor - 1$$

e preserva l'ordine.

- I casi più comuni, tuttavia, sono tali da richiedere l'uso di funzioni hash che si comportino bene per un'arbitraria distribuzione dei valori di chiave sul loro dominio di definizione. Nel seguito la discussione sarà limitata esclusivamente al caso di distribuzioni non note a priori.



Richiami (1)

Distribuzione geometrica

$$f(x) = pq^x \quad p = 1 - q$$

$$E[x] = \sum_{x=0}^{\infty} xf(x) = p \sum_{x=0}^{\infty} xq^x = pq \sum_{x=0}^{\infty} \frac{d}{dq} q^x =$$

$$= pq \frac{d}{dq} \sum_{x=0}^{\infty} q^x = pq \frac{d}{dq} \left(\frac{1}{1-q} \right) =$$

$$= pq \frac{1}{(1-q)^2} = \frac{q}{1-q}$$

Richiami (2)

Distribuzione binomiale

$$P(X = x) = \binom{n}{x} p^x q^{n-x}$$

n prove, x successi

p probabilità di successo

$q = 1-p$ probabilità di insuccesso

$$E[x] = np \quad \sigma^2 = npq$$

Esempio: 3 lanci di una moneta

CCC
CCT
CTC
~~CTT~~
TCC
~~TCT~~
~~TC~~
TTT

$$E[x] = \sum_{x=0}^n x \binom{n}{x} p^x q^{n-x} = \sum_{x=1}^n x \frac{n!}{x!(n-x)!} p^x q^{n-x} =$$

$$= \sum_{x=1}^n \frac{n!}{(x-1)!(n-x)!} p^x q^{n-x}$$

$$= np \sum_{x=1}^n \frac{(n-1)!}{(x-1)!(n-x)!} p^{x-1} q^{n-x}$$

ponendo $\xi = x-1$ $\eta = n-1$

$$E[x] = np \sum_{\xi=0}^{\eta} \frac{\eta!}{\xi!(\eta-\xi)!} p^{\xi} q^{\eta-\xi} = np$$

$$P(2 \text{ teste}) = \binom{3}{2} \left(\frac{1}{2}\right)^2 \left(\frac{1}{2}\right) = \frac{3}{8}$$

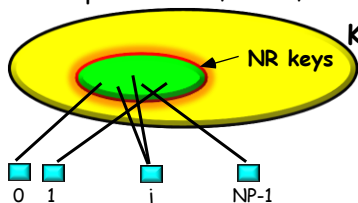
Organizzazioni dei dati- parte IV



15

Funzioni hash

- Una funzione hash è una trasformazione (**suriettiva**) dallo spazio, K , delle chiavi allo spazio, $\{0, \dots, NP - 1\}$, degli indirizzi. L'ipotesi che un **arbitrario** sottoinsieme di K si ripartisca sugli NP indirizzi in maniera perfettamente omogenea è una pura astrazione, di scarsa utilità per analizzare le prestazioni ottenibili dalle diverse organizzazioni hash.
- Il **caso ideale** rispetto al quale è ragionevole confrontare una specifica funzione hash H è quello di **distribuzione uniforme sullo spazio degli indirizzi**, in cui, per ogni sottoinsieme di K , ognuno degli NP indirizzi ha la stessa probabilità, $1/NP$, di essere generato.



Organizzazioni dei dati- parte IV



16



Funzioni hash: caso ideale

- Nel caso ideale il numero di chiavi, X_j , assegnate al bucket j-esimo segue una distribuzione binomiale.

$$\Pr\{X_j = x_j\} = \binom{NR}{x_j} \left(\frac{1}{NP}\right)^{x_j} \left(1 - \frac{1}{NP}\right)^{NR-x_j}$$

con valor medio μ e varianza σ^2 dati da:

$$\mu = \frac{NR}{NP} \quad \sigma^2 = \frac{NR}{NP} \times \left(1 - \frac{1}{NP}\right)$$

in cui né il valor medio né la varianza dipendono dallo specifico bucket.

- Per $NP \gg 1$, il rapporto $\sigma/\sqrt{\mu}$ vale circa 1.



Qualità di una funzione hash

- Nel caso di funzioni hash "reali" le prestazioni variano al variare dello specifico set di chiavi.
- Esempio:** La funzione $H(k_i) = k_i \bmod NP$ (**metodo della divisione**) è una "buona" funzione, ma nel caso, ad esempio, del set di chiavi: $\{NP, 2 \times NP, 3 \times NP, \dots, NR \times NP\}$ alloca tutte le chiavi nel **bucket 0**.
- Ogni funzione hash, scelta **indipendentemente** dallo specifico set di chiavi, può dar luogo a prestazioni disastrose nel caso peggiore. Nel caso "medio", tuttavia, considerando arbitrari sottoinsiemi di K e file dati reali, si osserva che le diverse funzioni hash si comportano effettivamente in modo diverso.
- Un criterio adeguato di valutazione di una funzione H , riferito a un particolare insieme di chiavi, è dato dall'analisi della sua **degenerazione**.

Degenerazione

- La degenerazione di una funzione H , riferita a uno specifico set di chiavi, è data dal rapporto:

$$\sigma_H / \sqrt{\mu_H}$$

dove

$$\mu_H = \sum_{j=0}^{NP-1} \frac{x_j}{NP} = \frac{NR}{NP} \quad \sigma_H^2 = \sum_{j=0}^{NP-1} \frac{(x_j - \mu_H)^2}{NP}$$

sono calcolati su tutti gli NP bucket e x_j è il numero di record osservato nel bucket j .

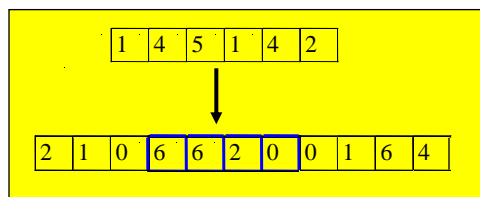
- Quanto **più bassa** è la degenerazione, tanto **migliore** è il comportamento della funzione hash.

Esempi di funzioni hash (1)

- Previa trasformazione di chiavi alfanumeriche in numeri, le più note funzioni hash sono:

MID SQUARE

La chiave è elevata al quadrato, si estrae poi un numero di cifre centrali pari a quelle di $(NP - 1)$, e il numero ottenuto è normalizzato a NP .



Se ad esempio $NP=8000$, la normalizzazione produce l'indirizzo:

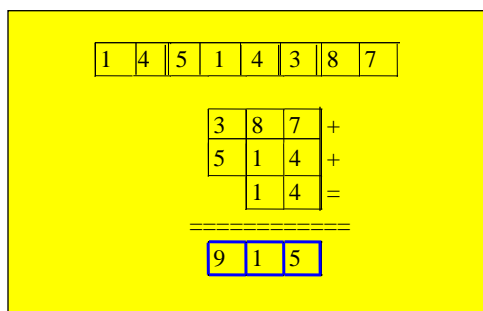
$$\lfloor 6620 \times 0.8 \rfloor = 5296$$



Esempi di funzioni hash (2)

SHIFTING

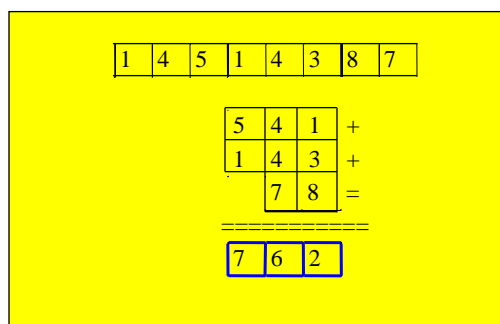
La chiave è suddivisa in un certo numero di parti, ognuna costituita da un numero di cifre pari a (numero di cifre di NP) - 1. Si sommano le parti e si normalizza il risultato.



Esempi di funzioni hash (3)

FOLDING

La chiave è suddivisa come nello shifting. Le parti vengono "ripiegate" (folded) e quindi sommate.



Esempi di funzioni hash (4)

DIVISIONE

La chiave numerica viene divisa per un numero P e l'indirizzo è ottenuto considerando il resto:

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

$$P = 6997$$

$$k = 172146 \quad H(k) = 4218$$

$$172147 \quad 4219$$

$$172148 \quad 4220$$

$$172149 \quad 4221$$

✚ per la scelta di P si hanno le seguenti indicazioni pratiche:

1. P è il più grande numero primo minore o uguale a NP;

2. P è non primo, minore o uguale a NP, con nessun fattore primo minore di 20.

✚ Se $P < NP$, si deve porre $NP := P$ per non perdere la suriettività della funzione hash.

Funzioni hash a confronto (1)

✚ Le prestazioni dei diversi metodi sono state analizzate con riferimento al numero di overflow generati, al variare della capacità dei bucket.

✚ I confronti sono eseguiti per diversi valori del **fattore di caricamento (load factor) d**, definito come il rapporto fra il numero di chiavi allocate e la capacità dell'area primaria:

$$d = \frac{NR}{NP \times C}$$

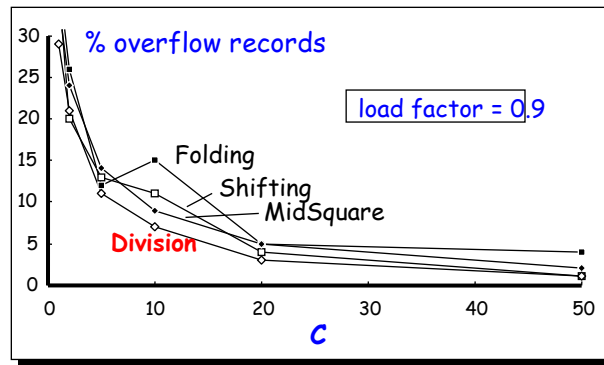
✚ Intuitivamente, all'aumentare del fattore di caricamento, aumenta la percentuale di record in overflow.

✚ d può essere maggiore di 1 nel caso di area separata per l'overflow.



Funzioni hash a confronto (1)

- Test sperimentali eseguiti su 8 files con caratteristiche molto diversificate mostrano che, in generale, il metodo della divisione è il più affidabile.



Funzioni hash 2-universali

- Analisi teoriche possono spiegare perché, in media, il metodo della divisione funziona bene (Carter e Wegman 1979 - funzioni hash n-universali).
- **Classi di funzioni hash 2-universali**

Dato un insieme K di chiavi $\{0, 1, \dots, |K| - 1\}$ e NP bucket, una classe $H = \{H_1, H_2, \dots\}$ di funzioni hash è 2-universale se, per ogni coppia di interi in K , il numero totale di collisioni tra le due chiavi in H è minore o uguale a $|H|/NP$.

Scegliendo a caso una funzione da H , la probabilità che due chiavi collidano è minore o uguale a $1/NP$. Nel caso di funzione hash ideale, la probabilità di collisione di due chiavi è proprio pari a $1/NP$. Qualitativamente, H contiene molte "buone" funzioni hash.

- La classe $H = \{((A \times k + D) \bmod P) \bmod NP \mid A \geq 1, D, P \geq 0\}$ è 2-universale. Ponendo $A=1$; $D=0$; $P = \infty$ si ottiene il metodo della divisione.



Chiavi alfanumeriche

- Il trattamento di stringhe alfanumeriche richiede una fase preliminare di conversione. Uno dei metodi più comuni consiste nello stabilire:
 - ✚ un alfabeto A , a cui appartengono i caratteri delle stringhe,
 - ✚ una funzione biiettiva $\text{ord}()$, che associa a ogni elemento dell'alfabeto un intero nel range $[1, |A|]$
 - ✚ una base di conversione b .
- Una stringa $S = s_{n-1}, \dots, s_i, \dots, s_0$ è quindi convertita in una chiave numerica:

$$k(S) = \sum_{i=0}^{n-1} \text{ord}(s_i) \times b^i$$



Esempio chiavi alfanumeriche

- Posto $A = \{a, b, \dots, z\}$, $\text{ord}()$ a valori in $[1, 26]$, e $b = 32$, la stringa "indice" produce il valore numerico:

$$\begin{aligned} k(\text{indice}) &= 9 \times 32^5 + 14 \times 32^4 + 4 \times 32^3 + \\ &\quad 9 \times 32^2 + 3 \times 32^1 + 5 \times 32^0 = 316,810,341 \end{aligned}$$

- Metodi più semplici che non fanno uso di una base funzionano meno bene; ad esempio

$$k_{(\text{bad})}(S) = \sum_{i=0}^{n-1} \text{ord}(s_i)$$

genera la stessa chiave numerica per due stringhe distinte che sono l'una anagramma dell'altra.

Per le stringhe $S_1 = \text{"volare"}$ e $S_2 = \text{"valore"}$ risulterebbe infatti $k_{(\text{bad})}(S_1) = k_{(\text{bad})}(S_2)$.



Scelta della base (1)

- Con il metodo della divisione si possono riscontrare seri problemi quando la base b e NP hanno fattori primi in comune.
- Sia $A = \{a, b, \dots, z\}$, $b = 32$ e $NP = 512$. Si può immediatamente verificare che il valore di $H(k(S))$ è determinato solo dagli ultimi due caratteri. Ad esempio:

"folder": i valori ordinali sono 6, 15, 12, 4, 5, 18 e

$$k(\text{folder}) = 217,452,722.$$

e l'indirizzo di bucket risulta pari a $H(217452722) = 178$

"primer": gli ordinali sono 16, 18, 9, 13, 5, 18,

$$k(\text{primer}) = 556,053,682$$

e l'indirizzo è ancora $H(556053682) = 178$



Scelta della base (2)

- ✚ Il fenomeno è causato dalle proprietà dell'operatore mod.
- ✚ Il caso più semplice da considerare è quello in cui NP è una potenza di b , cioè $NP = b^\alpha$. Allora esiste un valore i^* per cui $b^{i^*} \bmod (b^\alpha) = 0$. Poiché per l'operatore mod vale la proprietà (utile per il calcolo di $k(S)$):

$$\begin{aligned} H(k(S)) &= \left\{ \sum_{i=0}^{n-1} (\text{ord}(s_i) \times b^i) \right\} \bmod (b^\alpha) = \\ &= \left\{ \sum_{i=0}^{n-1} [(\text{ord}(s_i) \times b^i) \bmod (b^\alpha)] \right\} \bmod (b^\alpha) \end{aligned}$$

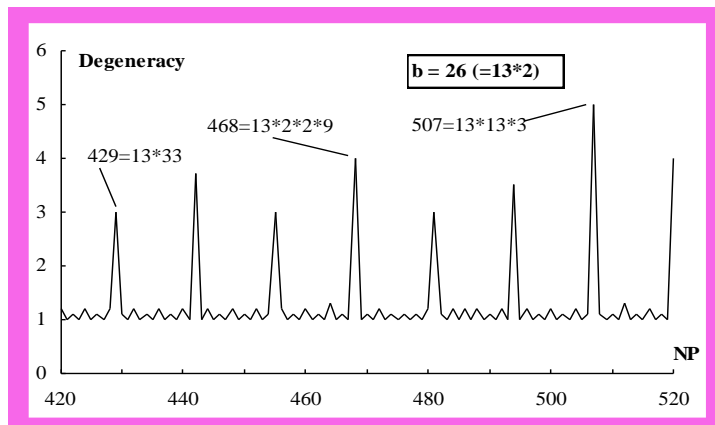
tutti i caratteri da s_{n-1} a s_{i^*} danno contributo nullo al valore di $H(k(S))$ (nell'esempio $i^* = 2$), e quindi la stringa "utile" è lunga solo i^* caratteri.

- ✚ Con base 26 si hanno problemi quando NP ha come fattori 13 e 2.



Esperimento di Mullin (1991)

- Insieme di chiavi: tutte le parole di 6 caratteri, costruite sull'alfabeto $A=\{a,b,\dots,z\}$, usate dallo spelling checker di Unix.
- $b \in \{26, 29, 32\}$; $NP \in \{420..520\}$.



Organizzazioni dei dati- parte IV

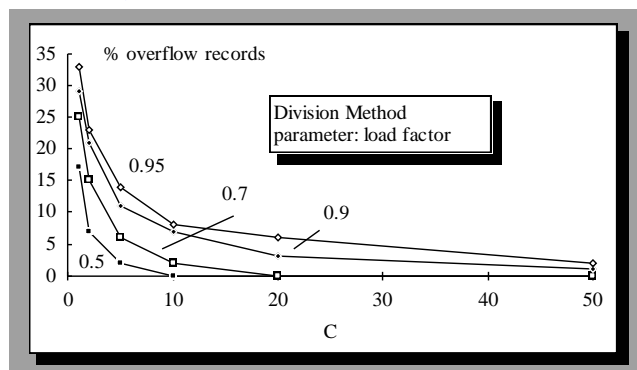


31



Fattore di caricamento (1)

- Data una stima del numero NR di record da gestire, e fissata la capacità C dei bucket, la scelta di un determinato fattore di caricamento, d , determina il numero di bucket, NP , in area primaria. *Va tenuto presente che, al diminuire di d , diminuisce la percentuale di record in overflow.*



Organizzazioni dei dati- parte IV

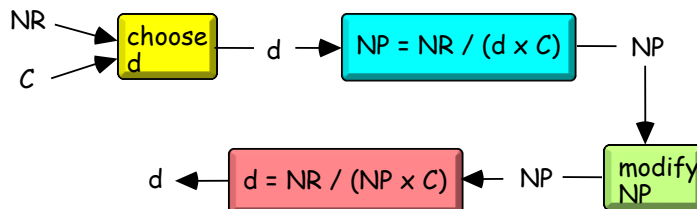


32



Fattore di caricamento (2)

- Per evitare problemi legati alla funzione hash, si può rendere necessario "ritoccare" il valore di NP, e, conseguentemente, quello di d.



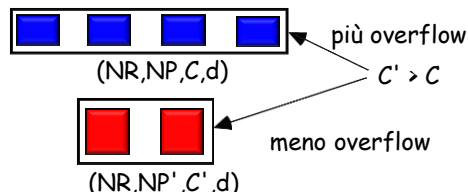
- Per ridurre la percentuale dei record in overflow, che incide sia sui costi di ricerca sia su quelli di aggiornamento, non è consigliabile utilizzare fattori di caricamento elevati. **Valori tipici**, che rappresentano un buon compromesso tra utilizzazione della memoria e costi di esecuzione delle operazioni, si hanno nell'intervallo **[0.7,0.8]**.



Capacità dei bucket

- Risulta conveniente avere bucket di capacità $C > 1$ a causa della relazione esistente tra C e la percentuale di record in overflow:

All'aumentare di C , e a parità di fattore di caricamento d , la percentuale di record in overflow diminuisce, sotto le ipotesi di una funzione hash ideale e di gestione degli overflow in area separata.



- Il risultato è valido anche nel caso di funzioni hash non ideali. Dunque è consigliabile scegliere C massima purché la lettura di un bucket comporti una singola operazione di I/O e il trasferimento di un bucket di capacità C avvenga in un tempo minore del trasferimento di due bucket di capacità minore di C .

Numero medio di overflow


C	d	100(NR _{OV} /NR)
1	0.5	21.31
	0.7	28.08
	0.9	34.06
	1.0	36.79
5	0.5	2.48
	0.7	7.11
	0.9	13.78
	1.0	17.55
10	0.5	0.44
	0.7	2.88
	0.9	8.59
	1.0	12.51
50	0.5	0
	0.7	0.05
	0.9	2.04
	1.0	5.63

$$NR_{OV}(C) \approx NP \times \sum_{x=C+1}^{NR} (x-C) \times \frac{(C \times d)^x e^{-C \times d}}{x!}$$

$$= NR \times \frac{(C \times d)^C e^{-C \times d}}{C!} \times f(C, d)$$

Nel caso di funzione hash ideale, il numero di volte che è generato l'indirizzo j è una variabile aleatoria che segue una distribuzione binomiale.

$$Pr(x) = \binom{NR}{x} \left(\frac{1}{NP}\right)^x \left(1 - \frac{1}{NP}\right)^{NR-x} \approx \left(\frac{NR}{NP}\right)^x \times \frac{e^{-NR/NP}}{x!}$$


 Organizzazioni dei dati- parte IV
 35

Gestione dell'overflow

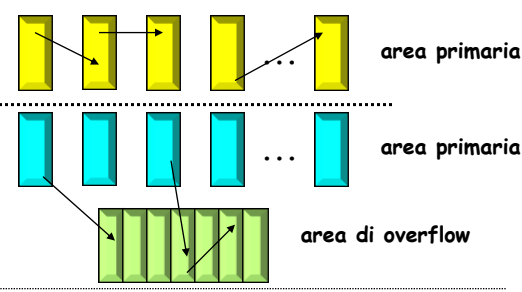
■ Obiettivo: ridurre al minimo il numero di accessi a bucket, necessari per reperire il record cercato. I diversi metodi si distinguono in:

Metodi di concatenamento (chaining)

fanno uso di puntatori; i record in overflow possono essere allocati in un'area separata o nella stessa area primaria

Metodi di indirizzamento aperto (open addressing)

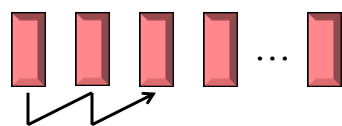
non fanno uso di puntatori; i record in overflow sono allocati in area primaria tramite una legge di scansione




area primaria

area primaria

area di overflow



area primaria


 Organizzazioni dei dati- parte IV
 36



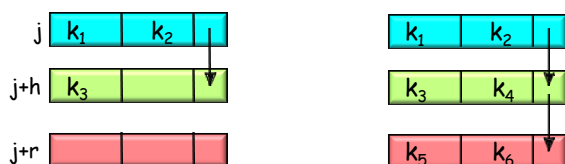
Chaining in area primaria

- **Liste separate:** se un bucket j è saturo, i record in overflow aventi j come **home bucket** sono allocati nel primo bucket non pieno, eseguendo una ricerca a partire dal bucket $j+1$. Tutti i record che collidono sono collegati a lista, inclusi quelli non in overflow. Ogni record deve pertanto includere un campo puntatore al record successivo della catena.
- **Liste confluenti (coalesced chaining):** a differenza del metodo precedente, si fa uso di un solo puntatore per bucket; se il bucket j va in overflow:
 - ✚ il record viene inserito nel primo bucket non pieno, $j + h$.
 - ✚ si attiva un collegamento da j a $j + h$
 - ✚ se anche $j + h$ va in overflow, allora i nuovi record vengono inseriti nel bucket $j + r$



Liste confluenti

- A causa degli overflow possono fondersi liste corrispondenti a home bucket diversi. Rispetto alla gestione con liste separate si semplifica la gestione dei puntatori, tuttavia peggiorano complessivamente le prestazioni.



$$H(k_1) = H(k_2) = H(k_3) = j$$

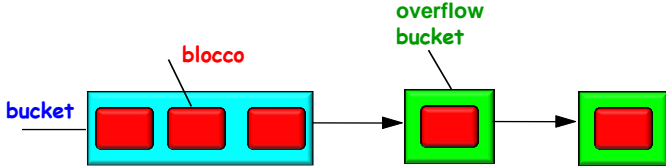
$$H(k_4) = j + h$$

$$H(k_5) = j$$

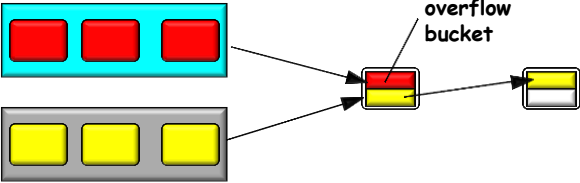
$$H(k_6) = j + r$$

Chaining in area separata


- Gli overflow sono memorizzati in un'area di memoria distinta da quella primaria, non indirizzata dalla funzione hash. La capacità C_{ov} dei bucket di overflow può essere minore di C , per evitare eccessivo spreco di spazio.



- Per lo stesso motivo è possibile anche che più bucket di overflow siano mappati in un singolo blocco.



i problemi di gestione dell'area di overflow sono simili a quelli della gestione di liste di TID nei posting file.

Organizzazioni dei dati- parte IV 

39


Open addressing (1)

- Nei metodi a indirizzamento aperto a ogni valore di chiave k_i viene associata una sequenza di indirizzi $H_0(k_i), H_1(k_i), \dots, H_l(k_i)$, con $H_0(k_i) = H(k_i)$. Lo schema generale per inserire un record con chiave k_i è:

```

{  $H_0(k_i) := H(k_i)$ ;           {home bucket}
   $l := 0$ ; FULL_AREA:=false;
  while FULL( $H_l(k_i)$ ) and (not FULL_AREA) do
  {bucket pieno}
    {  $l := l + 1$ ;
       $H_l(k_i) := STEP(H_{l-1}(k_i))$ ;
      {bucket successivo}
      FULL_AREA := evaluate( $H_l(k_i)$ );
      {bucket già esaminato?}
    }
    if (not FULL_AREA) then insert( $k_i, H_l(k_i)$ )
  }
```

- ✦ la funzione STEP determina il criterio di scansione dei bucket
- ✦ FULL($H_l(k_i)$) = true quando non vi sono posizioni "occupabili" nel bucket $H_l(k_i)$.
- ✦ FULL_AREA=true quando non si riesce a inserire la chiave

Organizzazioni dei dati- parte IV 

40



Open addressing (2)

- I metodi a indirizzamento aperto, così come quelli che non fanno uso di un'area di overflow separata, sono caratterizzati da un'utilizzazione della memoria allocata, u , che coincide con il fattore di caricamento ($u = d$), il quale deve pertanto sempre essere minore di, o al limite uguale a, 1.
- Nel caso in cui si renda necessario gestire un numero di record $NR > NP \times C$, si rende necessario provvedere a una riorganizzazione completa.
- Occorre prestare particolare attenzione alle operazioni di cancellazione; ciò dipende dal modo in cui è realizzata la ricerca di un record e dal significato che si attribuisce alla variabile **FULL**.



Open addressing (3)

Esempio

ordine di inserimento: 45, 16, 43, 38, 64, 33, 34, 89, 50

$$\text{STEP}(H_{i-1}(k_i)) = (H_{i-1}(k_i) + 1) \bmod NP$$

$$H(k) = k \bmod 5 \quad \text{○} = \text{overflow}$$

<div><div>34</div><div>45</div></div>	<div><div>89</div><div>16</div></div>	<div><div></div><div>50</div></div>	<div><div>38</div><div>43</div></div>	<div><div>33</div><div>64</div></div>
0	1	2	3	4

La cancellazione della chiave 34 non permetterebbe più di reperire 89 e 50; dunque si deve gestire lo stato di ogni posizione all'interno dei bucket e **FULL=false se e solo se esiste almeno una posizione "libera" nel bucket**.





Linear probing

- A ogni passo l'indirizzo è incrementato di una quantità costante s :

$$\text{STEP}(H_{l-1}(k_i)) = (H_{l-1}(k_i) + s) \bmod NP$$

cioè $H_l(k_i) = (H(k_i) + s \cdot l) \bmod NP$

- Per garantire che, per ogni home bucket, siano generati tutti gli NP indirizzi è necessario che s non abbia divisori in comune con NP . In caso contrario, sono generati solo $NP / \text{MCD}(NP, s)$ indirizzi distinti. Può pertanto accadere che non si riesca a inserire una chiave anche se il fattore di caricamento è minore di 1.

Esempio: se $NP = 10$ e $s = 4$, e quindi $\text{MCD}(10, 4) = 2$, allora dato, ad es., $H(k_i) = 3$, la sequenza consiste di soli 5 indirizzi distinti:

3, 7, 1, 5, 9, 3,



Linear probing: clustering primario

- Con la scansione lineare si osserva un fenomeno, detto di **clustering primario**, per cui i record tendono ad addensarsi in alcuni bucket, a causa della linearità di $\text{STEP}()$.

- **Esempio:** Sia $H(k_i) = k_i \bmod 31$; $s = 3$.

La chiave 1234 genera la sequenza

(25, 28, 0, 3, 6, 9, 12, ...)

La chiave 245 genera la sequenza

(28, 0, 3, 6, 9, 12, 15, ...)

Se il bucket 25 va in overflow, aumenta la probabilità che vada in overflow il 28, poi il bucket 0, ecc.



Scansione quadratica

- Il fenomeno di clustering primario può essere evitato se si fa in modo che le sequenze di indirizzi si diversifichino in funzione dell'home bucket. Ciò si ottiene facendo uso di una legge di **scansione quadratica**:

$$\text{STEP}(H_{l-1}(k_i)) = (H_{l-1}(k_i) + a + b(2 \times l - 1)) \bmod NP$$

ovvero: $H_l(k_i) = (H_0(k_i) + a \times l + b \times l^2) \bmod NP$

Esempio: con $a = 3$ e $b = 5$, le chiavi **1234** e **245** ora generano le sequenze **(25, 2, 20, 17, 24, 6, ...)** **(28, 5, 23, 20, 27, 13, ...)**; le sequenze non sono più confluenti, ad es. l'indirizzo successivo a 20 non è lo stesso per i due valori di chiave.

- La scansione quadratica non risolve tuttavia il problema del **clustering secondario**, dovuto a chiavi che hanno lo stesso home bucket, e che quindi produrranno sempre la stessa sequenza.



Double hashing

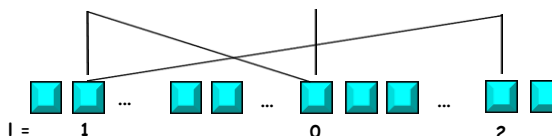
- Mira a eliminare i problemi dovuti al clustering secondario facendo uso di due funzioni hash, H' e H'' . Le sequenze di indirizzi sono date da:

$$H_0(k_i) = H'(k_i)$$

$$H_l(k_i) = (H_{l-1}(k_i) + H''(k_i)) \bmod NP \quad (l > 0)$$

- Due chiavi generano ora la stessa sequenza di indirizzi se e solo se collidono sia con H' sia con H'' . La tecnica di double hashing approssima abbastanza bene il caso ideale di "hash uniforme" (**random probing**), in cui ogni indirizzo ha la stessa probabilità di essere generato al passo l -esimo.

La tecnica di double hashing produce come effetto collaterale una forte variabilità degli indirizzi generati, dipendentemente dal valore di $H''(k_i)$. Considerando l'effettiva allocazione dei bucket in memoria secondaria ciò può appesantire notevolmente le operazioni di I/O.

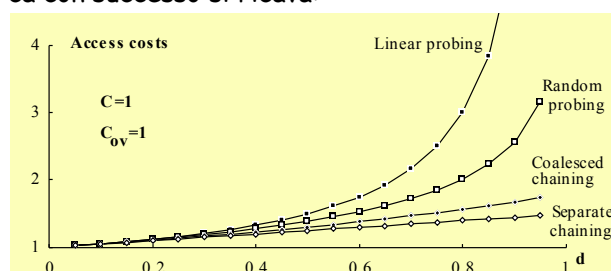


Prestazioni (1)

- Per i metodi che non fanno uso di area separata di overflow si ha un'utilizzazione $u=d$. Facendo uso di bucket di overflow:

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

dove NP_{ov} è il numero di bucket di overflow allocati. Per i costi medi di ricerca con successo si ricava:



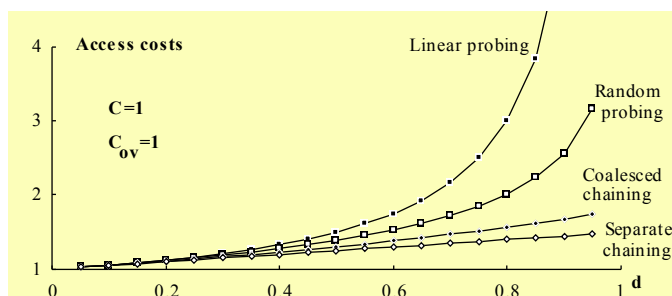
Organizzazioni dei dati- parte IV



47

Prestazioni (2)

- Per i costi medi di ricerca con insuccesso le prestazioni peggiorano notevolmente per i metodi open addressing:



Organizzazioni dei dati- parte IV



48

Prestazioni metodi di chaining

- L'analisi, basata sull'ipotesi di disporre di funzioni hash ideali, porta ai seguenti risultati approssimati, derivati per bucket di capacità unitaria e validi per $NP \gg 1$ ($NP \rightarrow \infty$). Si indica con E ("Exists") il costo medio di ricerca con successo, e con A ("Absent") il costo medio di ricerca con insuccesso.

✚ Coaleshed chaining

$$E \approx 1 + \frac{1}{8 \times d} (e^{2 \times d} - 1 - 2 \times d) + \frac{d}{4}$$

$$A \approx 1 + \frac{1}{4} (e^{2 \times d} - 1 - 2 \times d)$$

- ✚ **Separate chaining** (ulteriore ipotesi $C_{ov}=1$, record non ordinati per valore di chiave nelle liste di overflow)

$$E \approx 1 + \frac{1}{2 \times d}$$

$$A \approx e^{-d} + d$$

Organizzazione dei record nei bucket

- La scelta di una politica di allocazione dei record nei bucket influisce sui costi delle diverse operazioni e/o sull'occupazione di memoria.

