



Università degli Studi "G.d'Annunzio" Chieti – Pescara
Corso di Laurea in Economia Informatica

Algoritmi e Strutture di Dati 2 – A.A. 2002/2003
Dott. Massimo Franceschet – Prof.ssa Maria Chiara Meo

Introduzione alle tabelle hash

di Enrico Zimuel
(enrico@zimuel.it)

revisione del 22/3/2003



Sommario

- L'idea delle tabelle hash
- Tabelle ad indirizzamento diretto
- Tabelle hash
- Risoluzione delle collisioni: concatenazione, indirizzamento aperto
- Analisi dell'organizzazione hash con concatenazione
- Requisiti di una buona funzione hash
- Esempi di funzioni hash: metodo di divisione e di moltiplicazione
- Funzione hash universale
- Indirizzamento aperto: scansione lineare, quadratica, hashing doppio
- Analisi dell'organizzazione hash con indirizzamento aperto



L'idea delle tabelle hash

- Una tabella hash è una struttura di dati dinamica efficace per realizzare i *dizionari*.
- Un *dizionario* è una collezione **S** di elementi, cui sono associate chiavi **k** (ciascuno contenuto al più una volta). Su un *dizionario* sono definite le seguenti operazioni: Inserimento (**Insert**), Ricerca (**Search**), Eliminazione (**Delete**).
- Per esempio, un compilatore di un linguaggio di programmazione mantiene una tabella di simboli **S**, in cui le chiavi **k** degli elementi sono stringhe di caratteri qualunque che corrispondono a identificatori del linguaggio.
- Una tabella hash è la generalizzazione del più semplice concetto di array ordinario.

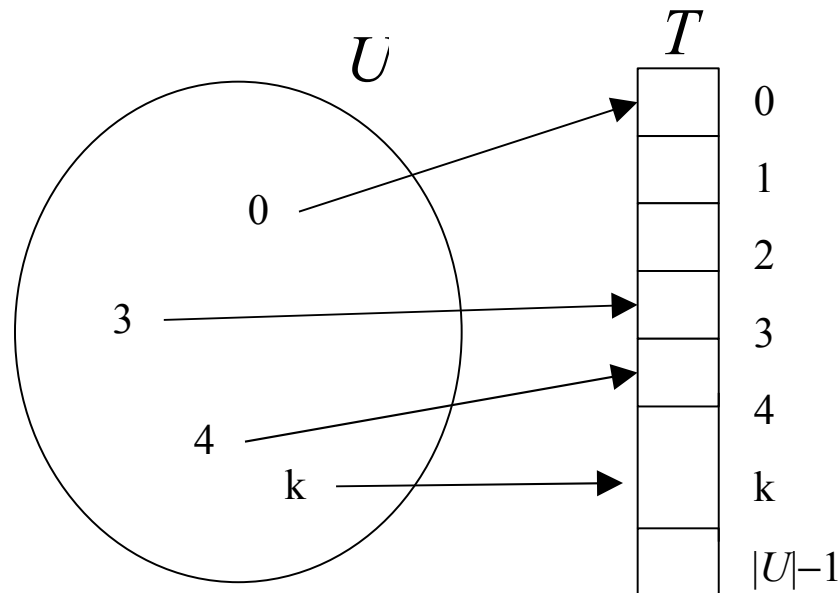


L'idea delle tabelle hash

- L'utilizzo di un array ordinario può essere applicato quando ci si possa permettere di allocare un array con una posizione per ogni possibile chiave.
- Quando il numero delle chiavi è piccolo rispetto al numero totale di possibili chiavi, le tabelle hash diventano un'alternativa efficace all'indirizzamento diretto di un array.
- L'idea delle tabelle hash: invece di usare la chiave come indice per indirizzare direttamente l'array, l'indice è **calcolato** utilizzando la chiave. Il calcolo dell'indice è delegato ad una funzione hash.
- Vedremo che in media il tempo di ricerca di un elemento in una tabella hash è $O(1)$.

Tabelle ad indirizzamento diretto

- L'indirizzamento diretto è una tecnica semplice che funziona bene quando l'universo \mathbf{U} delle chiavi è ragionevolmente piccolo.
- Per rappresentare l'insieme dinamico, si usa un array, o *tabella ad indirizzamento diretto*, $\mathbf{T}[0.. m-1]$ in cui ogni posizione, o *slot*, corrisponde ad una chiave nell'universo \mathbf{U} .



La posizione k punta ad un elemento dell'insieme con chiave k . Se l'insieme non contiene elementi con chiave k , allora $T[k]=\text{nil}$.



Tabelle ad indirizzamento diretto

- Le operazioni fondamentali su di un *dizionario* con una tabella ad indirizzamento diretto sono banali da realizzare:

Direct-Address-Search (T,k)

return $T[k]$

Direct-Address-Insert (T,x)

$T[\text{key}[x]] \leftarrow x$

Direct-Address-Delete(T,x)

$T[\text{key}[x]] \leftarrow \text{nil}$

- La complessità computazionale di queste operazioni è $O(1)$.



Tabelle hash

- La difficoltà dell'indirizzamento diretto è legata alla dimensione di U : se l'universo U è grande, memorizzare una tabella T di dimensione $|U|$ può essere impraticabile, o addirittura impossibile.
- Memorizziamo, ad esempio, il vocabolario della lingua italiana con l'utilizzo di una tabella ad indirizzamento diretto. L'insieme U delle possibili chiavi è enorme, poiché per ogni parola del vocabolario deve essere associata una chiave che garantisca la dinamicità della struttura. Un modo per calcolare la chiave k può essere il seguente: alla stringa $a_{n-1} \dots a_0$ è associata la chiave calcolata in base 26:

$$k = \text{key}[a_{n-1} \dots a_0] = \sum_{i=0}^{n-1} n(a_i) \cdot b^i \quad \text{con} \quad n(a_i) \in \{1 \dots 26\} \quad \text{e} \quad b = 26$$

Ad esempio la parola "zuccherificio" ha come chiave k il valore $26 \cdot 26^{12} + 21 \cdot 26^{11} + 3 \cdot 26^{10} + 3 \cdot 26^9 + \dots + 15 \cdot 26^0 > 2,4$ miliardi di miliardi (250 mila TB – TeraByte = 2^{40} byte)



Tabelle hash

- Altra considerazione: l'insieme delle chiavi **K** effettivamente utilizzate può essere così piccolo rispetto ad **U** che la maggior parte dello spazio allocato per **T** sarebbe inutilizzato.
- Si definisce una tabella hash **T**[0..m-1] ed funzione hash tra l'universo **U** di tutte le possibili chiavi e le posizioni della tabella hash **h**: $U \rightarrow \{0,1,\dots,m-1\}$, $h(k)$ è il *valore hash* di k .
- Il punto essenziale della funzione hash è di ridurre l'intervallo degli indici dell'array che devono essere gestiti. Invece di $|U|$ valori, si devono gestire solo m valori.
- Con una tabella hash la memoria richiesta può essere ridotta a $\Theta(|K|)$.



Tabelle hash

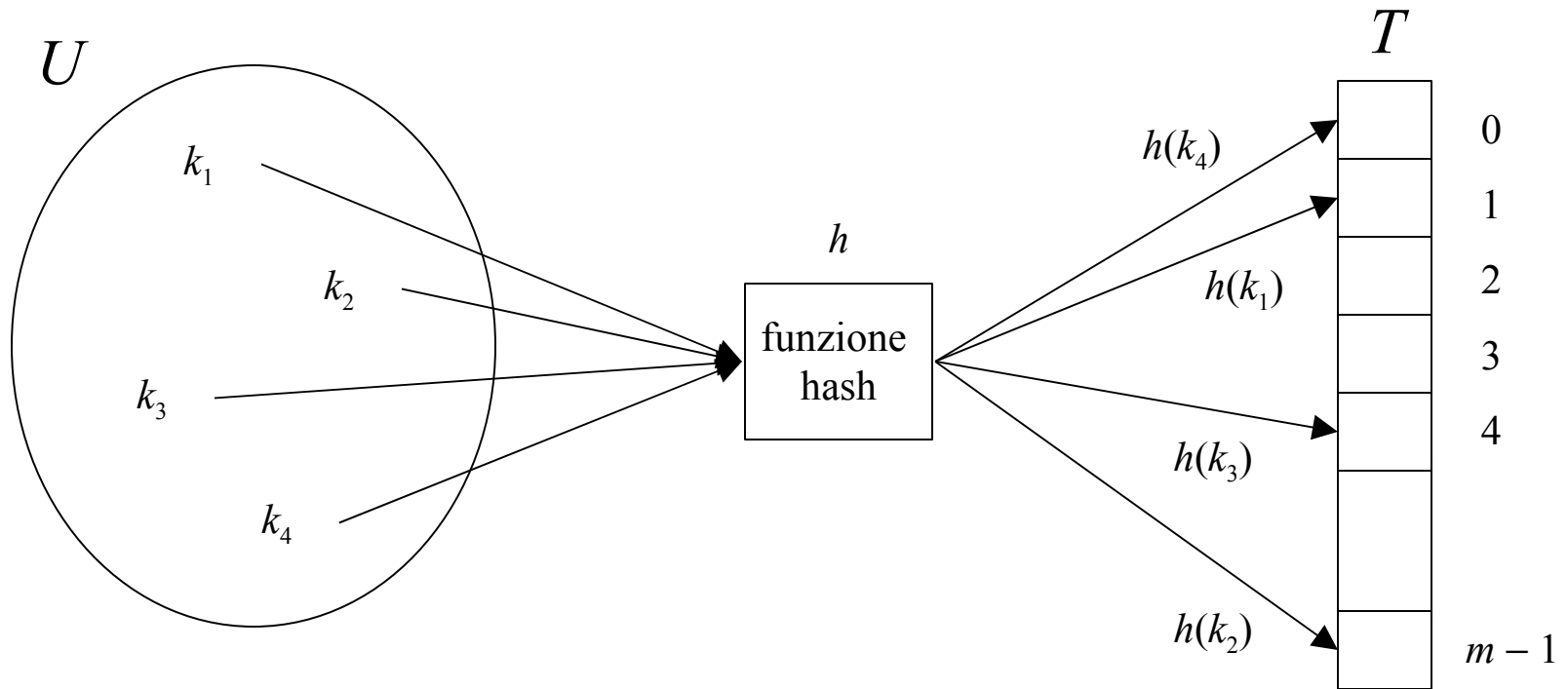


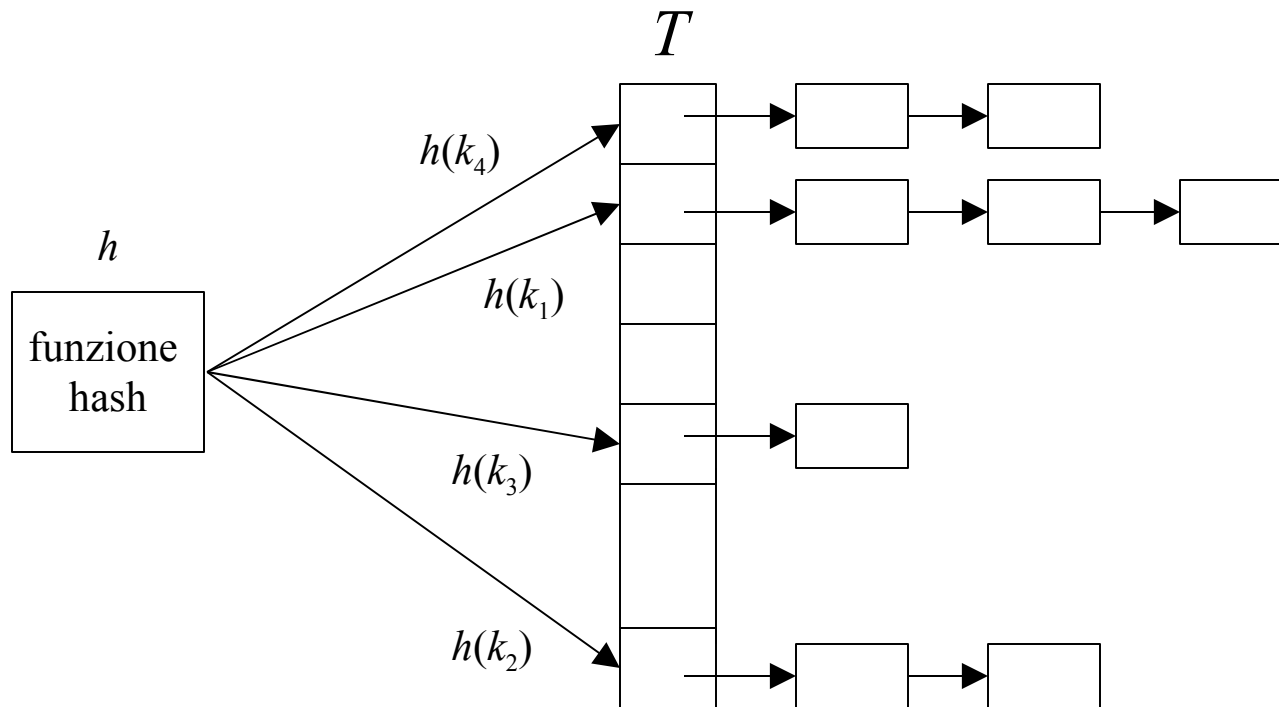


Tabelle hash

- Due chiavi k_i e k_j con $i \neq j$ possono corrispondere alla stessa posizione $h(k_i) = h(k_j)$, questo fenomeno è chiamato *collisione*.
- Per diminuire la probabilità delle collisioni dobbiamo scegliere la funzione hash $h(k)$ in modo che generi un intervallo il più possibile "casuale" (il termine inglese hash ha diversi significati tra i quali *tagliare*, *fare a pezzetti*, *incasinare* ed indica anche "il nome che si dà al polpettone o più precisamente al piatto di carne avanzata del giorno prima e tritata insieme con verdure")
- Poiché $|U| > m$ vi sono sicuramente due chiavi con lo stesso valore hash, evitare del tutto le collisioni è quindi impossibile.
- Non ci resta che gestire le collisioni.

Risoluzione delle collisioni per concatenazione

La tabella hash \mathbf{T} è un array di puntatori a liste: l'elemento x tale che $\text{key}[x]=k$ viene inserito nella lista puntata da $\mathbf{T}[\mathbf{h}(k)]$.





Risoluzione delle collisioni per concatenazione

- Le operazioni fondamentali sul dizionario implementate con una tabella hash T quando le collisioni sono risolte per concatenazione risultano:

Chained-Hash-Insert (T,x)

inserisci x in testa alla lista $T[h(\text{key}[x])]$

Chained-Hash-Search (T,k)

ricerca un elemento con chiave k nella lista $T[h(k)]$

Chained-Hash-Delete (T,x)

cancella x dalla lista $T[h(\text{key}[x])]$

- Insert: $O(1)$ nel caso peggiore, Search: $O(a)$ dove a è la lunghezza della lista nel caso peggiore, Delete: $O(1)$ nel caso peggiore con liste bidirezionali, $O(a)$ nel caso di liste semplici



Analisi dell'organizzazione hash con concatenazione

- Data una tabella hash **T** con **m** posizioni che memorizza **n** elementi si definisce il **fattore di carico** $\alpha = n/m$, cioè il numero medio di elementi memorizzati in ogni lista concatenata.
- Il comportamento medio dell'organizzazione hash dipende da quanto, in media, la funzione hash distribuisca bene l'insieme di chiavi da memorizzare sulle **m** posizioni.
- **Ipotesi di uniformità semplice della funzione hash:** qualunque elemento corrisponde in modo equamente probabile a una delle **m** posizioni, indipendentemente dalla posizione degli altri elementi.
- Si assume che il valore hash $h(k)$ possa essere calcolato in tempo $O(1)$, così che il tempo richiesto per cercare un elemento con chiave **k** dipenda in modo lineare dalla lunghezza della lista $T[h(k)]$.



Analisi dell'organizzazione hash con concatenazione

• Teorema 1

In una tabella hash in cui le collisioni sono risolte per concatenazione, nell'ipotesi di uniformità semplice della funzione hash, una ricerca senza successo richiede in media tempo $\Theta(1+\alpha)$

Dimostrazione:

Ipotesi di uniformità semplice della funzione hash \Rightarrow equiprobabilità uniforme sulle m posizioni. Il tempo medio della ricerca senza successo di una chiave k è dato dalla lunghezza media della lista $T[h(k)]$ che è pari al fattore di carico $\alpha=n/m$. Dunque il tempo totale richiesto è pari alla somma del tempo di calcolo di $h(k)$ e della scansione della lista $T[h(k)]$, ossia è $\Theta(1+\alpha)$



Analisi dell'organizzazione hash con concatenazione

• Teorema 2

In una tabella hash in cui le collisioni sono risolte per concatenazione, nell'ipotesi di uniformità semplice della funzione hash, una ricerca con successo richiede in media tempo $\Theta(1+\alpha)$

Dimostrazione:

Ipotesi di uniformità semplice della funzione hash \Rightarrow equiprobabilità uniforme sulle m posizioni. Si assume che la procedura Chained-Hash-Insert inserisca un nuovo elemento in coda alla lista piuttosto che in testa (la complessità della ricerca non cambia e assumendo l'utilizzo di liste circolari anche la complessità dell'inserimento non cambia). Per trovare il numero medio di elementi esaminati si prende la media, sugli n elementi della tabella, di 1 più la lunghezza media della lista in cui l'elemento i -esimo è aggiunto. La lunghezza media della lista è $(i-1)/m$ per cui il numero medio di elementi esaminati si può calcolare in questo modo:



Analisi dell'organizzazione hash con concatenazione

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n \left(1 + \frac{i-1}{m} \right) &= \frac{1}{n} \sum_{i=1}^n \left(\frac{m+i-1}{m} \right) = \frac{1}{nm} \sum_{i=1}^n (m+i-1) = \\ &= \frac{1}{nm} \left(nm + \sum_{i=1}^n (i-1) \right) = 1 + \frac{1}{nm} \sum_{i=1}^n (i-1) = 1 + \frac{1}{nm} \left(\frac{n(n-1)}{2} \right) = \end{aligned}$$

Utilizzo la formula di Gauss

$$\sum_{i=1}^n (i-1) = \sum_{k=0}^{n-1} k = \sum_{k=1}^{n-1} k = \frac{(n-1+1)(n-1)}{2} = \frac{n(n-1)}{2}$$

$$= 1 + \frac{n-1}{2m} = 1 + \frac{n}{2m} - \frac{1}{2m} = 1 + \frac{\alpha}{2} - \frac{1}{2m} = 1 + \frac{\alpha}{2} - \frac{\alpha}{2n} = 1 + \frac{\alpha}{2} \left(1 - \frac{1}{n} \right) = 1 + c \frac{\alpha}{2} \quad 0 \leq c < 1$$



Analisi dell'organizzazione hash con concatenazione

Per cui $\Theta(1+c(\alpha/2))=\Theta(1+\alpha)$ con $0 \leq c < 1$.

- Da questa analisi se ne deduce che se il numero di posizioni nella tabella hash è almeno proporzionale al numero di elementi nella tabella, ossia $n=O(m)$, si ha $\alpha=n/m=O(m)/m=O(1)$. Quindi la ricerca richiede in media tempo costante.
- Poiché l'inserzione richiede tempo $O(1)$ nel caso peggiore e la cancellazione, quando le liste sono bidirezionali, richiede tempo $O(1)$ nel caso peggiore, tutte le operazioni del dizionario possono essere eseguite in media con tempo $O(1)$.



Requisiti di una buona funzione hash

- Una buona funzione hash deve essere semplice da calcolare ($O(1)$) e deve soddisfare, con una certa approssimazione, l'ipotesi di uniformità semplice. In pratica la probabilità che una chiave k abbia come valore hash un numero compreso tra 0 ed $m-1$ deve essere costante e pari a $1/m$:

$$P(h(k)=j) = \frac{1}{m} \quad \forall j=0,1,\dots,m-1$$

- Dal punto di vista pratico non è possibile controllare questa condizione poiché la distribuzione di probabilità P di solito non è conosciuta.
- Si procede in maniera euristica per la creazione di una funzione hash che rispetti le proprietà precedenti. Un approccio comune consiste nel derivare il valore hash in modo che sia indipendente da qualunque configurazione possa esistere dei dati.



Calcolo di una funzione hash: il metodo di divisione

- Il metodo di divisione fa corrispondere una chiave k ad una delle m posizioni prendendo il resto della divisione di k per m :

$$h(k) = k \bmod m$$

- Bisogna fare attenzione nella scelta del valore di m poiché ci possono essere dei valori numerici che non rendono uniforme la distribuzione delle chiavi k . Ad esempio m non dovrebbe essere una potenza di 2, poiché se $m=2^p$ allora $h(k)$ è dato dai p bit meno significativi di k .
- Buoni valori di m sono valori primi non troppo vicini a potenze esatte di 2. Ad esempio si supponga di voler allocare una tabella hash, con concatenazione, per contenere $n=2000$ stringhe di caratteri. Esaminare in media 3 elementi in una ricerca senza successo può essere soddisfacente, così si sceglie $m=701$ ($\alpha = 2000/701 \approx 2,85$).



Calcolo di una funzione hash: il metodo di moltiplicazione

- Il metodo di moltiplicazione per il calcolo della funzione hash opera in due passi: prima si moltiplica la chiave k per una costante A con $0 < A < 1$ e si estrae la parte frazionaria di kA , poi si moltiplica questo valore per m e si prende la parte intera del risultato:

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

dove $kA \bmod 1$ rappresenta la parte frazionaria di kA , cioè $kA - \lfloor kA \rfloor$.

- Un vantaggio del metodo di moltiplicazione è che il valore di m non è critico come per il metodo di divisione. Tipicamente si sceglie una potenza di 2.
- La scelta della costante A influenza la qualità della funzione hash, la scelta ottimale dipende dalle caratteristiche dei dati su cui vengono prese le chiavi.



Funzione hash universale

- Lavorando con chiavi differenti ci può essere l'inconveniente di generare n chiavi che corrispondano tutte alla stessa posizione provocando un tempo di ricerca $\Theta(n)$.
- Per ovviare a questo problema si deve scegliere una funzione hash in *modo casuale* cosicché sia indipendente dalle chiavi che vanno effettivamente memorizzate.
- Questa tecnica è chiamata **hashing universale**.
- L'idea di base di quest'approccio è di selezionare, in modo casuale, la funzione hash al tempo di esecuzione da una classe di funzioni attentamente definite. A causa della scelta casuale, l'algoritmo si può comportare diversamente ad ogni esecuzione, anche per lo stesso input.



Funzione hash universale

- Sia **H** un insieme finito di funzioni hash che vanno da un dato universo **U** di chiavi all'intervallo $\{0,1,\dots,m-1\}$.
- **H** è detto **universale** se per ogni coppia di chiavi distinte $x,y \in U$ il numero di funzioni hash $h \in H$ per cui $h(x)=h(y)$ è precisamente $|H|/m$. In altre parole, con una funzione hash scelta in modo casuale da H la probabilità di una collisione tra x e y per $x \neq y$ è esattamente $1/m$ che è esattamente la probabilità di una collisione se $h(x)$ e $h(y)$ sono scelte casualmente nell'insieme $\{0,1,\dots,m-1\}$.
- Vale il seguente **Teorema 3**:
Se h è scelta da un insieme universale di funzioni hash ed è usata per la corrispondenza di n chiavi su una tabella di dimensione m , gestendo le collisioni con la tecnica della concatenazione valgono i teoremi 1 e 2 dimostrati in precedenza.



Funzione hash universale

- Quanto è semplice definire una classe universale di funzioni hash?
- Si può utilizzare questo algoritmo: si scelga come dimensione **m** della tabella un numero primo. Si decompone una chiave x in $r+1$ byte, così che $x = \langle x_0, x_1, \dots, x_r \rangle$; il massimo valore di un byte deve essere minore di m . Si denoti con $a = \langle a_0, a_1, \dots, a_r \rangle$ una sequenza di $r+1$ elementi scelti a caso dall'insieme $\{0, 1, \dots, m-1\}$. Si definisce una funzione hash $h_a \in H$:

$$h_a(x) = \sum_{i=0}^r a_i x_i \text{ mod } m \quad (1.1)$$

$$H = \bigcup_a \{h_a\} \quad (1.2)$$

con queste definizioni $|H| = m^{r+1}$



Funzione hash universale

- Vale il seguente teorema:

Teorema 4

La classe H definita dalle equazioni (1.1) e (1.2) è una classe universale di funzioni hash.



Risoluzione delle collisioni: indirizzamento aperto

- Nell'*indirizzamento aperto* tutti gli elementi sono memorizzati nella tabella hash stessa. Cioè ogni elemento della tabella contiene o un elemento dell'insieme dinamico o NIL.
- Il vantaggio dell'*indirizzamento aperto* è di evitare del tutto i puntatori, invece di seguire i puntatori, come nella concatenazione, si *calcola* la sequenza di posizioni da esaminare.
- Per determinare la posizione da esaminare, si estende la funzione hash perché includa il numero di posizioni già esaminate (che parte da 0) come secondo input. La funzione hash risulta dunque:

$$h: U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}$$



Risoluzione delle collisioni: indirizzamento aperto

- Con l'indirizzamento aperto si richiede che per ogni chiave k , la sequenza di scansione

$\langle h(k,0), h(k,1), \dots, h(k,m-1) \rangle$

sia una permutazione di $\langle 0,1,\dots,m-1 \rangle$ così che ogni elemento della tabella hash sia eventualmente considerato come una posizione per la nuova chiave fino a riempire tutta la tabella.



Risoluzione delle collisioni: indirizzamento aperto

- Nel seguente pseudocodice si assume che la chiave k è proprio l'elemento stesso della tabella T che la contiene. Ogni posizione contiene o una chiave o NIL (se la posizione è vuota).

Hash-Insert (T, k)

```
i ← 0
repeat
    j ← h(k,i)
    if T[j]=NIL then
        T[j] ← k
        return j
    else i ← i+1
until i=m
error "overflow sulla tabella hash"
```



Risoluzione delle collisioni: indirizzamento aperto

Hash-Search (T,k)

```
i ← 0
repeat
  j ← h(k,i)
  if T[j]=k then
    return j
  i ← i+1
until (T[j]=NIL) or (i=m)
return NIL
```



Risoluzione delle collisioni: indirizzamento aperto

- L'operazione di cancellazione da una tabella hash ad indirizzamento aperto è difficile. Quando si cancella una chiave da una posizione i , non si può semplicemente marcare quella posizione come vuota tramite NIL.
- Così facendo potrebbe essere impossibile recuperare qualunque chiave k per la quale, durante l'inserzione, la posizione i era stata esaminata e trovata occupata.
- Una soluzione è di marcare la posizione memorizzandovi il valore speciale DELETED invece di NIL. Si dovrebbe allora modificare la procedura Hash-Insert per prevedere questa possibilità. Così facendo, però, i tempi di ricerca non dipendono più soltanto dal fattore di carico α . Per questa ragione, quando le chiavi devono essere cancellate, di solito si sceglie la tecnica di risoluzione delle collisioni per concatenazione.



Risoluzione delle collisioni: indirizzamento aperto

- Nell'analisi del metodo dell'indirizzamento aperto si fa l'ipotesi della **uniformità della funzione hash**: ogni chiave considerata ha, in modo equiprobabile, una qualunque delle $m!$ permutazioni di $\{0,1,\dots,m-1\}$ come sequenza di scansione ($m!$ scansioni) .
- L'uniformità della funzione hash generalizza la nozione di uniformità semplice, definita in precedenza, alla situazione in cui la funzione hash produce non un singolo valore ma un'intera sequenza di scansione.
- In verità una funzione hash uniforme è difficile da realizzare e in pratica vengono utilizzate delle approssimazioni. Noi ne esamineremo tre: la scansione lineare, quadratica e l'hashing doppio.



Indirizzamento aperto: scansione lineare

- **Scansione lineare**

Data una funzione hash $h': U \rightarrow \{0,1,\dots,m-1\}$, il metodo di scansione lineare usa la funzione:

$$h(k,i) = (h'(k) + i) \bmod m$$

per $i=0,1,\dots,m-1$.

- La scansione realizza m scansioni per ogni chiave k .
- Fenomeno di ***agglomerazione primaria***. Le posizioni occupate si accumulano in lunghi tratti (di m posizioni), aumentando il tempo medio di ricerca. La scansione lineare non è una buona approssimazione di una funzione hash uniforme.



Indirizzamento aperto: scansione quadratica

- **Scansione quadratica**

La scansione quadratica usa una funzione hash della forma:

$$h(k,i) = (h'(k) + c_1i + c_2i^2) \bmod m$$

dove h' è una funzione hash ausiliaria, c_1 e $c_2 \neq 0$ sono costanti e $i=0,1,\dots,m-1$.

- La scansione realizza m scansioni per ogni chiave k .
- Fenomeno di ***agglomerazione secondaria***. Come per la scansione lineare, l'accesso iniziale determina l'intera sequenza, quindi sono usate solo m distinte sequenze di scansione. Anche se rispetto all'agglomerazione primaria la situazione è migliorata. La scansione quadratica, così come la lineare, non è una buona approssimazione di una funzione hash uniforme.



Indirizzamento aperto: hashing doppio

- **Hashing doppio**

L'hashing doppio è uno dei migliori metodi disponibili per l'indirizzamento aperto perché le permutazioni prodotte hanno molte delle caratteristiche delle permutazioni scelte casualmente. La funzione di hashing doppio è la seguente:

$$h(k,i) = (h_1(k) + i h_2(k)) \bmod m$$

dove h_1 e h_2 sono funzioni hash ausiliarie.

- Un modo per scegliere l'hashing doppio potrebbe essere il seguente: si sceglie m primo e si pone $h_1(k) = k \bmod m$ e $h_2(k) = 1 + (k \bmod m')$ dove m' è scelto in modo che sia di poco minore di m (ad esempio $m-1$ o $m-2$).
- L'hashing doppio rappresenta un miglioramento della scansione lineare o quadratica poiché realizza m^2 scansioni.



Analisi dell'organizzazione hash con indirizzamento aperto

- Valgono i seguenti teoremi:

Teorema 5

Data una tabella hash a indirizzamento aperto con fattore di carico $\alpha = n/m < 1$, il numero medio di accessi di una ricerca senza successo è al più $1/(1 - \alpha)$, assumendo l'uniformità della funzione hash.

Dimostrazione

Si denota con p_i la probabilità di avere esattamente i collisioni, il costo di una ricerca con insuccesso è:

$$1 + \sum_{i=1}^{\infty} ip_i$$

Dove il termine 1 è il costo per l'accesso all'unica posizione vuota dell'array che non dà collisione, e $p_i = 0$ per $i > n$, essendoci n chiavi memorizzate.



Analisi dell'organizzazione hash con indirizzamento aperto

Per valutare la serie precedente si usa la relazione:

$$\sum_{i=1}^{\infty} ip_i = \sum_{i=1}^{\infty} q_i$$

Dove q_i denota la probabilità di avere almeno i collisioni. Per ricavare la probabilità si osservi che $q_1 = n/m = \alpha$, perché si ha almeno una collisione accedendo ad una delle n posizioni occupate sulle m totali. Proseguendo con il calcolo:

$$q_2 = \frac{n}{m} \left(\frac{n-1}{m-1} \right) \quad \dots \quad q_i = \frac{n}{m} \prod_{k=1}^i \frac{n-k}{m-k}$$



Analisi dell'organizzazione hash con indirizzamento aperto

Per m ed n grandi, quest'ultima quantità si può maggiore con $(n/m)^i = \alpha^i$. Pertanto il costo di una ricerca con insuccesso è, per $0 < \alpha < 1$:

$$C(\alpha) \leq 1 + \sum_{i=1}^{\infty} \alpha^i = \frac{1}{1-\alpha}$$

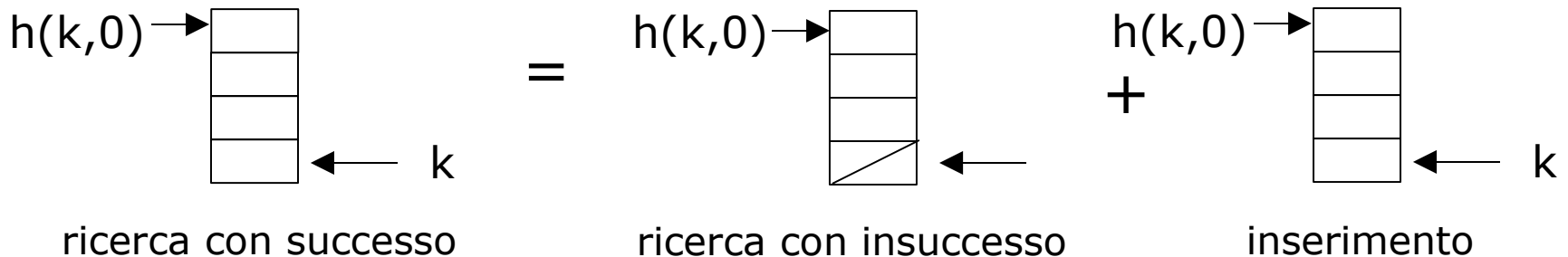
Analisi dell'organizzazione hash con indirizzamento aperto

Teorema 6

Data una tabella hash ad indirizzamento aperto con fattore di carico $\alpha < 1$, il numero medio di accessi in una ricerca con successo è al più $(1/\alpha)\ln(1/(1-\alpha))$ nell'ipotesi di una funzione hash uniforme e assumendo che ogni chiave sia ricercata nella tabella in modo equiprobabile.

Dimostrazione

Il costo di una ricerca con successo è pari al costo di una ricerca con insuccesso quando la chiave è stata inserita, poiché in entrambi i casi viene effettuata la stessa sequenza di scansione.





Analisi dell'organizzazione hash con indirizzamento aperto

Al momento dell'inserzione il fattore di carico è diverso da quello che si ha al momento della ricerca con successo. Sia β il fattore di carico all'atto dell'inserzione ed α quello all'atto della ricerca con successo. Facendo la media sui possibili valori di β , che sono compresi tra 0 ed α , si ottiene:

$$C(\alpha) \leq \frac{1}{\alpha} \int_0^{\alpha} \frac{1}{1-\beta} d\beta = \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

La disuguaglianza (\leq) dell'espressione precedente dipende dal fatto che α e β sono numeri razionali e non reali.

Il numero medio di accessi in una ricerca con successo risulta essere al più $(1/\alpha)\ln(1/(1-\alpha))$.



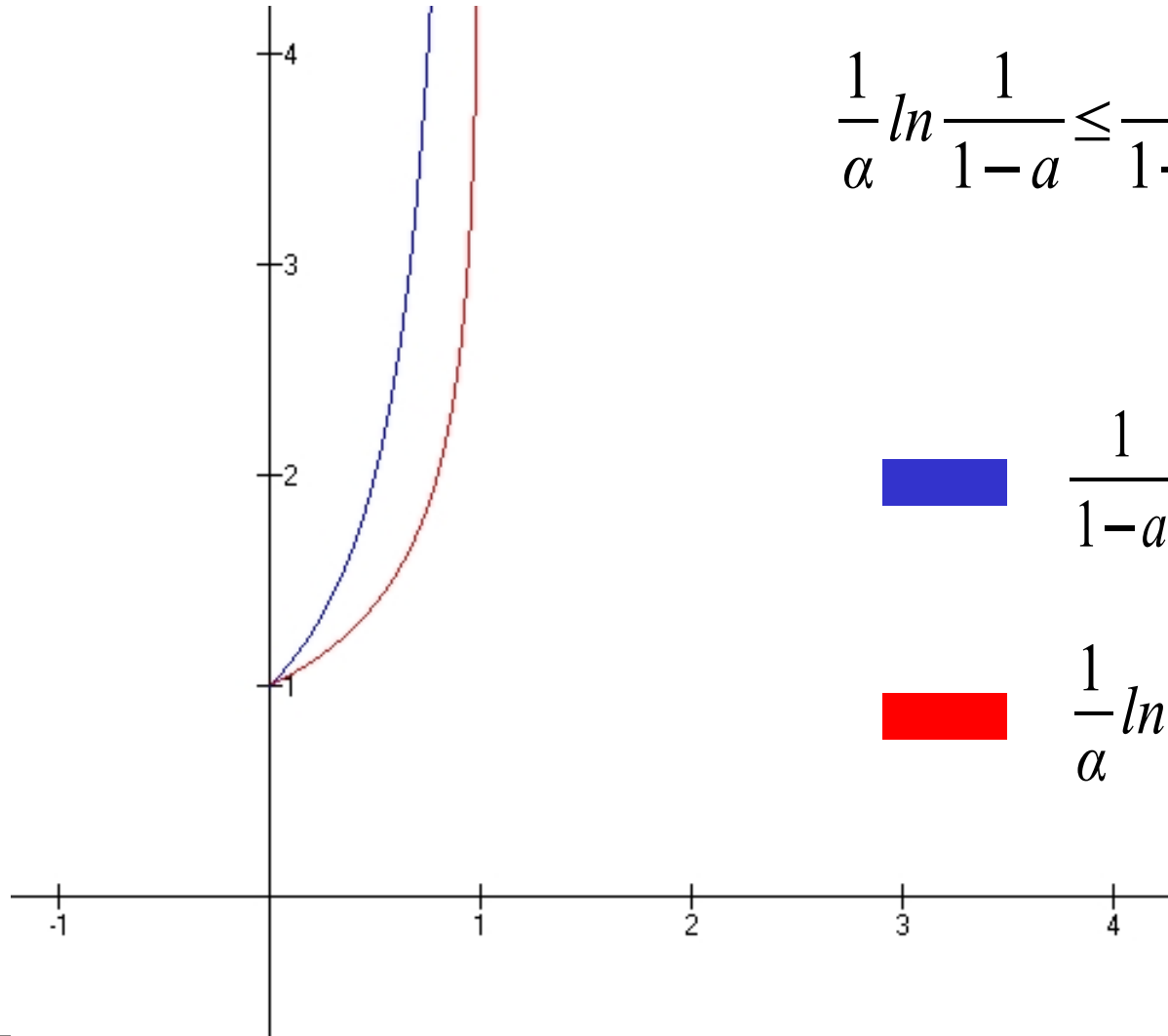
Analisi dell'organizzazione hash con indirizzamento aperto

- Osservando semplicemente l'algoritmo di ricerca Hash-Search(T,k) se ne deduce che la complessità di una ricerca con successo è minore della complessità della ricerca con insuccesso.
- Dal punto di vista analitico la stessa osservazione non risulta così ovvia, poiché $(1/\alpha) > 1$:


$$\frac{1}{\alpha} \ln \frac{1}{1-a} \leq \frac{1}{1-a}$$




Analisi dell'organizzazione hash con indirizzamento aperto



$$\frac{1}{\alpha} \ln \frac{1}{1-a} \leq \frac{1}{1-a}$$

 $\frac{1}{1-a}$

 $\frac{1}{\alpha} \ln \frac{1}{1-a}$



Bibliografia

- "Introduction to Algorithms – Second edition" Cormen, Leiserson, Rivest, Stein (MIT Press, 2001) pag. 221-252
- "Algoritmi e strutture di dati" Bertossi Alan A. (Utet edizioni, 2000) pag. 93-109
- "Sorting and Searching – Vol.3 The Art of Computer Programming" Donald E.Knuth (Addison-Wesley, 1973)
- "Handbook of Algorithms and Data Structure" G.H.Gonnet (Addison-Wesley, 1984)
- "Algoritmi, divinità e gente comune" Luccio Fabrizio, Pagli Linda (Ets edizioni, 1999)