Algoritmi e Strutture Dati
Tabelle Hash

P. Massazza[1]

[1]Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese Italy

Outline

Outline

Algebre eterogenee

Ricordiamo l'equazione di N. Wirth

Programmi = Algoritmi + Strutture di Dati

e formalizziamo l'equazione

Struttura di Dati = Insiemi + Operazioni

attraverso la nozione di

Definizione [algebra eterogenea]

$A = \langle [A_1, \ldots, A_n], [f_1, \ldots, f_k] \rangle$ dove

- A_i sono insiemi;
- $f_j : A_{c_{j1}} \times \cdots \times A_{c_{jr_j}} \mapsto A_{i_j}, \quad c_{pq}, i_j \in \{1, \ldots, n\}$

Algebre eterogenee

Esempio

1. Parti di $A = \{a_1, \ldots, a_m\}$:

$$\text{PA} = \langle [A, 2^A, \text{Boolean}], [\text{Member}, \text{Insert}, \text{Delete}] \rangle$$

$$\text{Member} : A \times 2^A \mapsto \text{Boolean}, \text{Insert}, \text{Delete} : A \times 2^A \mapsto 2^A$$

2. (Parti di A ordinato)

$$\text{PAO} = \langle [A, 2^A, \text{Boolean}], [\text{Member}, \text{Insert}, \text{Delete}, \text{Min}] \rangle$$

come PA, in aggiunta $\text{Min} : 2^A \mapsto A$

Algebre eterogenee

Esempio

1. Parti di $A = \{a_1, \ldots, a_m\}$:

   $$\text{PA} = \langle [A, 2^A, \text{Boolean}], [\text{Member}, \text{Insert}, \text{Delete}] \rangle$$

   $\text{Member} : A \times 2^A \mapsto \text{Boolean}$, $\text{Insert}, \text{Delete} : A \times 2^A \mapsto 2^A$
2. (Parti di A ordinato)

   $$\text{PAO} = \langle [A, 2^A, \text{Boolean}], [\text{Member}, \text{Insert}, \text{Delete}, \text{Min}] \rangle$$

   come PA, in aggiunta $\text{Min} : 2^A \mapsto A$

Parti di A

Studiamo possibili implementazioni per la struttura dati PA

- Liste concatenate
  - Vantaggi: semplicità
  - Svantaggi: costo medio di ciascuna operazione $O(n)$
- Tabelle hash
  - Vantaggi: costo medio di ciascuna operazione $O(1)$
  - Svantaggi: costo di di ciascuna operazione $O(n)$ se si sbaglia a impostare alcuni parametri

Parti di A

Studiamo possibili implementazioni per la struttura dati PA

- Liste concatenate

  Vantaggi: semplicità
  Svantaggi: costo medio di ciascuna operazione $O(n)$

- Tabelle hash

  Vantaggi: costo medio di ciascuna operazione $O(1)$
  Svantaggi: costo di di ciascuna operazione $O(n)$ se si sbaglia a impostare alcuni parametri

Parti di A

Studiamo possibili implementazioni per la struttura dati PA

- Liste concatenate

  Vantaggi: semplicità
  Svantaggi: costo medio di ciascuna operazione $O(n)$

- Tabelle hash

  Vantaggi: costo medio di ciascuna operazione $O(1)$
  Svantaggi: costo di di ciascuna operazione $O(n)$ se si sbaglia a impostare alcuni parametri

Outline

Tabelle Hash

Una tabella Hash costituisce un'implementazione valida per PA. Distinguiamo tra

Hash statico la tabella ha dimensione fissa

Hash dinamico la dimensione della tabella varia dinamicamente

In funzione di dove sono memorizzati i dati si presentano 2 metodi di gestione

Concatenazioni separate: i dati risiedono in liste concatenate (la tabella contiene dei riferimenti)

Indirizzamento aperto: i dati risiedono all'interno della tabella

Abbiamo quindi 4 casi possibili...

Tabelle Hash

Una tabella Hash costituisce un'implementazione valida per PA. Distinguiamo tra

Hash statico la tabella ha dimensione fissa

Hash dinamico la dimensione della tabella varia dinamicamente

In funzione di dove sono memorizzati i dati si presentano 2 metodi di gestione

Concatenazioni separate: i dati risiedono in liste concatenate (la tabella contiene dei riferimenti)

Indirizzamento aperto: i dati risiedono all'interno della tabella

Abbiamo quindi 4 casi possibili...

Tabelle Hash

Una tabella Hash costituisce un'implementazione valida per PA. Distinguiamo tra

Hash statico la tabella ha dimensione fissa

Hash dinamico la dimensione della tabella varia dinamicamente

In funzione di dove sono memorizzati i dati si presentano 2 metodi di gestione

Concatenazioni separate: i dati risiedono in liste concatenate (la tabella contiene dei riferimenti)

Indirizzamento aperto: i dati risiedono all'interno della tabella

Abbiamo quindi 4 casi possibili...

Tabelle Hash: regole del gioco

- Ogni dato è contraddistinto da una chiave $c \in U$ che determina una posizione della tabella
- il numero di possibili chiavi è elevato, $k = \theta(2^{|c|})$
- Ogni posizione di una tabella di dimensione M è identificata da un indirizzo $i \in \{0, \ldots, M-1\}$
- $M \ll k$
- Una funzione di Hash H trasforma chiavi in indirizzi,

$$H : U \mapsto \{0, \ldots, M-1\}$$

Tabelle Hash: regole del gioco

- Ogni dato è contraddistinto da una chiave $c \in U$ che determina una posizione della tabella
- il numero di possibili chiavi è elevato, $k = \theta(2^{|c|})$
- Ogni posizione di una tabella di dimensione M è identificata da un indirizzo $i \in \{0, \ldots, M-1\}$
- $M \ll k$
- Una funzione di Hash H trasforma chiavi in indirizzi,

$$H : U \mapsto \{0, \ldots, M-1\}$$

Tabelle Hash: regole del gioco

- Ogni dato è contraddistinto da una chiave $c \in U$ che determina una posizione della tabella
- il numero di possibili chiavi è elevato, $k = \theta(2^{|c|})$
- Ogni posizione di una tabella di dimensione M è identificata da un indirizzo $i \in \{0, \ldots, M-1\}$
- $M \ll k$
- Una funzione di Hash H trasforma chiavi in indirizzi,

$$H : U \mapsto \{0, \ldots, M-1\}$$

Tabelle Hash: regole del gioco

- Ogni dato è contraddistinto da una chiave $c \in U$ che determina una posizione della tabella
- il numero di possibili chiavi è elevato, $k = \theta(2^{|c|})$
- Ogni posizione di una tabella di dimensione M è identificata da un indirizzo $i \in \{0, \ldots, M-1\}$
- $M \ll k$
- Una funzione di Hash H trasforma chiavi in indirizzi,

$$H : U \mapsto \{0, \ldots, M-1\}$$

Funzioni di Hash

La funzione di Hash viene utilizzata per calcolare la posizione di un dato nella tabella e deve avere le seguenti caratteristiche

1. facile da calcolare (costo $O(|c|)$)
2. distribuire uniformemente le chiavi su M indirizzi,

$$\forall c \in U, \forall i \in \{0, \ldots, M-1\}, \quad P(H(c) = i) = \frac{1}{M}$$

3. l'indirizzo calcolato deve dipendere da tutti i bit di c

Funzioni di Hash

La funzione di Hash viene utilizzata per calcolare la posizione di un dato nella tabella e deve avere le seguenti caratteristiche

1. facile da calcolare (costo $O(|c|)$)
2. distribuire uniformemente le chiavi su M indirizzi,

$$\forall c \in U, \forall i \in \{0, \ldots, M-1\}, \quad P(H(c) = i) = \frac{1}{M}$$

3. l'indirizzo calcolato deve dipendere da tutti i bit di c

Funzioni di Hash

La funzione di Hash viene utilizzata per calcolare la posizione di un dato nella tabella e deve avere le seguenti caratteristiche

1. facile da calcolare (costo $O(|c|)$)
2. distribuire uniformemente le chiavi su M indirizzi,

$$\forall c \in U, \forall i \in \{0, \ldots, M-1\}, \quad P(H(c) = i) = \frac{1}{M}$$

3. l'indirizzo calcolato deve dipendere da tutti i bit di c

Funzioni di Hash

La funzione di Hash viene utilizzata per calcolare la posizione di un dato nella tabella e deve avere le seguenti caratteristiche

1. facile da calcolare (costo $O(|c|)$)
2. distribuire uniformemente le chiavi su M indirizzi,

$$\forall c \in U, \forall i \in \{0, \ldots, M-1\}, \quad P(H(c) = i) = \frac{1}{M}$$

3. l'indirizzo calcolato deve dipendere da tutti i bit di c

Funzioni di Hash

Esempio

- La funzione identità $H(c) = c$ soddisfa 1,2,3 ma non è utilizzabile poiché $M \ll k$
- La funzione costante $H(c) = r$ soddisfa 1 ma non 2,3

Una possibile soluzione:
Considerare ogni chiave c come un intero di $|c|$ bit e prendere il resto della divisione per M

$$H(c) = \langle c \rangle_M$$

La scelta di M risulta critica perché la condizione 2 sia soddisfatta

Funzioni di Hash

Esempio

- La funzione identità $H(c) = c$ soddisfa 1,2,3 ma non è utilizzabile poiché $M \ll k$
- La funzione costante $H(c) = r$ soddisfa 1 ma non 2,3

Una possibile soluzione:
Considerare ogni chiave c come un intero di $|c|$ bit e prendere il resto della divisione per M

$$H(c) = \langle c \rangle_M$$

La scelta di M risulta critica perché la condizione 2 sia soddisfatta

Funzioni di Hash

Esempio

- La funzione identità $H(c) = c$ soddisfa 1,2,3 ma non è utilizzabile poiché $M \ll k$
- La funzione costante $H(c) = r$ soddisfa 1 ma non 2,3

Una possibile soluzione:
Considerare ogni chiave c come un intero di $|c|$ bit e prendere il resto della divisione per M

$$H(c) = \langle c \rangle_M$$

La scelta di M risulta critica perché la condizione 2 sia soddisfatta

Funzioni di Hash Modulari

Se le chiavi sono stringhe su un alfabeto di R caratteri le interpretiamo come interi in base R.

Esempio

Considerando la codifica ASCII a 7 bit ($R = 128$) :

- la chiave "ora" corrisponde all'intero

$$111 \cdot 128^2 + 114 \cdot 128^1 + 97 \cdot 128^0 = 1833313$$

Se $M = 128$ il valore dipende solo dall'ultimo carattere!
Infatti, $H(\text{ora}) = \langle 1833313 \rangle_{128} = 97$

- la chiave "averylongkey" ha un valore intero troppo grande

$$97 \cdot 128^{11} + 118 \cdot 128^{10} + \ldots, +121 \cdot 128^0 > 2^{7^{11}} = 2^{77}$$

Funzioni di Hash Modulari

Se le chiavi sono stringhe su un alfabeto di R caratteri le interpretiamo come interi in base R.

Esempio

Considerando la codifica ASCII a 7 bit ($R = 128$) :

- la chiave "ora" corrisponde all'intero

$$111 \cdot 128^2 + 114 \cdot 128^1 + 97 \cdot 128^0 = 1833313$$

Se $M = 128$ il valore dipende solo dall'ultimo carattere!
Infatti, $H(\text{ora}) = \langle 1833313 \rangle_{128} = 97$

- la chiave "averylongkey" ha un valore intero troppo grande

$$97 \cdot 128^{11} + 118 \cdot 128^{10} + \ldots, +121 \cdot 128^0 > 2^{7^{11}} = 2^{77}$$

Funzioni di Hash Modulari

Se le chiavi sono stringhe su un alfabeto di R caratteri le interpretiamo come interi in base R.

Esempio

Considerando la codifica ASCII a 7 bit ($R = 128$) :

- la chiave "ora" corrisponde all'intero

$$111 \cdot 128^2 + 114 \cdot 128^1 + 97 \cdot 128^0 = 1833313$$

Se $M = 128$ il valore dipende solo dall'ultimo carattere!
Infatti, $H(\text{ora}) = \langle 1833313 \rangle_{128} = 97$

- la chiave "averylongkey" ha un valore intero troppo grande

$$97 \cdot 128^{11} + 118 \cdot 128^{10} + \ldots, +121 \cdot 128^0 > 2^{7^{11}} = 2^{77}$$

Funzioni di Hash Modulari

Se le chiavi sono stringhe su un alfabeto di R caratteri le interpretiamo come interi in base R.

Esempio

Considerando la codifica ASCII a 7 bit ($R = 128$) :

- la chiave "ora" corrisponde all'intero

$$111 \cdot 128^2 + 114 \cdot 128^1 + 97 \cdot 128^0 = 1833313$$

  Se $M = 128$ il valore dipende solo dall'ultimo carattere!
  Infatti, $H(\text{ora}) = \langle 1833313 \rangle_{128} = 97$

- la chiave "averylongkey" ha un valore intero troppo grande

$$97 \cdot 128^{11} + 118 \cdot 128^{10} + \ldots, +121 \cdot 128^0 > 2^{7^{11}} = 2^{77}$$

Funzioni di Hash Modulari

- Per calcolare in maniera efficiente $H(c)$ e risolvere il problema dell'overflow sfruttiamo le proprietà dell'aritmetica modulare

$$\begin{aligned} \langle A + B \rangle_M &= \langle \langle A \rangle_M + \langle B \rangle_M \rangle_M \\ \langle A \cdot B \rangle_M &= \langle \langle A \rangle_M \cdot \langle B \rangle_M \rangle_M \end{aligned}$$

e utilizziamo la regola di Horner

$$a_0 x^n + a_1 x^{n-1} + a_n x^0 = a_n + x(a_{n-1} + x(a_{n-2} + \ldots (a_1 + x a_0) \ldots)))$$

- Per fare in modo che ogni bit della chiave influisca sull'indirizzo calcolato, M, R devono essere primi fra loro (M primo)

Funzioni di Hash Modulari

- Per calcolare in maniera efficiente $H(c)$ e risolvere il problema dell'overflow sfruttiamo le proprietà dell'aritmetica modulare

$$\begin{aligned} \langle A + B \rangle_M &= \langle \langle A \rangle_M + \langle B \rangle_M \rangle_M \\ \langle A \cdot B \rangle_M &= \langle \langle A \rangle_M \cdot \langle B \rangle_M \rangle_M \end{aligned}$$

  e utilizziamo la regola di Horner

$$a_0 x^n + a_1 x^{n-1} + a_n x^0 = a_n + x(a_{n-1} + x(a_{n-2} + \ldots (a_1 + x a_0) \ldots)))$$

- Per fare in modo che ogni bit della chiave influisca sull'indirizzo calcolato, M, R devono essere primi fra loro (M primo)

Funzioni di Hash Modulari

- Per calcolare in maniera efficiente $H(c)$ e risolvere il problema dell'overflow sfruttiamo le proprietà dell'aritmetica modulare

$$\begin{aligned} \langle A + B \rangle_M &= \langle \langle A \rangle_M + \langle B \rangle_M \rangle_M \\ \langle A \cdot B \rangle_M &= \langle \langle A \rangle_M \cdot \langle B \rangle_M \rangle_M \end{aligned}$$

  e utilizziamo la regola di Horner

$$a_0 x^n + a_1 x^{n-1} + a_n x^0 = a_n + x(a_{n-1} + x(a_{n-2} + \ldots (a_1 + x a_0) \ldots)))$$

- Per fare in modo che ogni bit della chiave influisca sull'indirizzo calcolato, M, R devono essere primi fra loro (M primo)

Funzioni di Hash Modulari

Esempio

H(averylongkey) viene calcolato come segue

$$97 \cdot 128^{11} + 118 \cdot 128^{10} + \ldots, +121 \cdot 128^{0} =$$
$$121 + 128(101 + 128(\ldots + 128(118 + 97 \cdot 128)\ldots)$$

```
public int Hash(String v)
{int h,l,m; char c;
 l=v.length(); m=0;
 for(h=1;h<=l;h++){
  c=v.charAt(h);
  m=(DIMALFA*m+getNumericValue(c))%DIMTAB}
 return m;
}
```

Funzioni di Hash Modulari

Esempio

H(averylongkey) viene calcolato come segue

$$97 \cdot 128^{11} + 118 \cdot 128^{10} + \ldots, +121 \cdot 128^{0} =$$
$$121 + 128(101 + 128(\ldots + 128(118 + 97 \cdot 128)\ldots)$$

```
public int Hash(String v)
{int h,l,m; char c;
 l=v.length(); m=0;
 for(h=1;h<=l;h++){
  c=v.charAt(h);
  m=(DIMALFA*m+getNumericValue(c))%DIMTAB}
 return m;
}
```

Funzioni di Hash Modulari

Esempio

H(averylongkey) viene calcolato come segue

$$97 \cdot 128^{11} + 118 \cdot 128^{10} + \ldots, +121 \cdot 128^{0} =$$
$$121 + 128(101 + 128(\ldots + 128(118 + 97 \cdot 128)\ldots)$$

```
public int Hash(String v)
{int h,l,m; char c;
 l=v.length(); m=0;
 for(h=1;h<=l;h++){
  c=v.charAt(h);
  m=(DIMALFA*m+getNumericValue(c))%DIMTAB}
 return m;
}
```

Funzioni di Hash: collisioni

Una funzione di Hash è necessariamente molti a uno. Occorre quindi gestire le collisioni, ovvero quando

$$x, y \in U, x \neq y, \quad H(x) = H(y)$$

La tecnica utilizzata dipende se operiamo con

- concatenazioni separate (soluzione immediata)
- indirizzamento aperto (soluzione più complessa)

Funzioni di Hash: collisioni

Una funzione di Hash è necessariamente molti a uno. Occorre quindi gestire le collisioni, ovvero quando

$$x, y \in U, x \neq y, \quad H(x) = H(y)$$

La tecnica utilizzata dipende se operiamo con

- concatenazioni separate (soluzione immediata)
- indirizzamento aperto (soluzione più complessa)

Funzioni di Hash: collisioni

Una funzione di Hash è necessariamente molti a uno. Occorre quindi gestire le collisioni, ovvero quando

$$x, y \in U, x \neq y, \quad H(x) = H(y)$$

La tecnica utilizzata dipende se operiamo con

- concatenazioni separate (soluzione immediata)
- indirizzamento aperto (soluzione più complessa)

Hash statico con concatenazioni separate

Si utilizza una Tabella T di dimensione fissa M. Per ogni i, $0 \leq i < M$, $T[i]$ contiene il riferimento al primo nodo della lista contenente dati con chiavi x tali che $H(x) = i$.

Member(x) si scorre la lista riferita da $H(x.chiave)$ fino a quando si trova x

Insert(x) si effettua un inserimento in testa in $H(x.chiave)$

Delete(x) si scorre la lista riferita da $H(x.chiave)$ fino a quando si trova x e lo si cancella

Hash statico con concatenazioni separate

Si utilizza una Tabella T di dimensione fissa M. Per ogni i, $0 \leq i < M$, $T[i]$ contiene il riferimento al primo nodo della lista contenente dati con chiavi x tali che $H(x) = i$.

Member(x) si scorre la lista riferita da $H(x.chiave)$ fino a quando si trova x

Insert(x) si effettua un inserimento in testa in $H(x.chiave)$

Delete(x) si scorre la lista riferita da $H(x.chiave)$ fino a quando si trova x e lo si cancella

Hash statico con concatenazioni separate

Si utilizza una Tabella T di dimensione fissa M. Per ogni i, $0 \leq i < M$, $T[i]$ contiene il riferimento al primo nodo della lista contenente dati con chiavi x tali che $H(x) = i$.

Member(x) si scorre la lista riferita da $H(x.chiave)$ fino a quando si trova x

Insert(x) si effettua un inserimento in testa in $H(x.chiave)$

Delete(x) si scorre la lista riferita da $H(x.chiave)$ fino a quando si trova x e lo si cancella

Hash statico con concatenazioni separate

Si utilizza una Tabella T di dimensione fissa M. Per ogni i, $0 \leq i < M$, $T[i]$ contiene il riferimento al primo nodo della lista contenente dati con chiavi x tali che $H(x) = i$.

Member(x)	si scorre la lista riferita da $H(x.chiave)$ fino a quando si trova x
Insert(x)	si effettua un inserimento in testa in $H(x.chiave)$
Delete(x)	si scorre la lista riferita da $H(x.chiave)$ fino a quando si trova x e lo si cancella

Hash statico con concatenazioni separate

Prestazioni

Se

- la funzione di Hash rispetta le condizioni 1,2 e 3
- T ha dimensione M e contiene n dati

Allora la lunghezza media di ciascuna lista sarà

$$\frac{n}{M}$$

Note

- ciascuna operazione ha costo pari a quello di un'operazione su lista (con fattore moltiplicativo $\frac{1}{M}$)
- Se abbiamo una stima sul numero di dati attesi otteniamo un costo medio $O(1)$ scegliendo $M \approx n$

Hash statico con concatenazioni separate

Prestazioni

Se

- la funzione di Hash rispetta le condizioni 1,2 e 3
- T ha dimensione M e contiene n dati

Allora la lunghezza media di ciascuna lista sarà

$$\frac{n}{M}$$

Note

- ciascuna operazione ha costo pari a quello di un'operazione su lista (con fattore moltiplicativo $\frac{1}{M}$)
- Se abbiamo una stima sul numero di dati attesi otteniamo un costo medio $O(1)$ scegliendo $M \approx n$

Hash statico con concatenazioni separate

Prestazioni

Se

- la funzione di Hash rispetta le condizioni 1,2 e 3
- T ha dimensione M e contiene n dati

Allora la lunghezza media di ciascuna lista sarà

$$\frac{n}{M}$$

Note

- ciascuna operazione ha costo pari a quello di un'operazione su lista (con fattore moltiplicativo $\frac{1}{M}$)
- Se abbiamo una stima sul numero di dati attesi otteniamo un costo medio $O(1)$ scegliendo $M \approx n$

Hash statico con concatenazioni separate

Teorema

Sia $\alpha = \frac{N}{M}$. La probabilità che una data lista di una tabella

- sia lunga k è minore di

$$\frac{\alpha^k e^{-\alpha}}{k!}$$

- abbia più di $t\alpha$ dati è minore di

$$\left(\frac{\alpha e}{t}\right)^t e^{-\alpha}$$

Ad esempio, se $\alpha = 20$ la probabilità di osservare una lista con almeno 40 elementi è minore di

$$\left(\frac{20e}{2}\right)^2 e^{-20} \approx 0.0000016$$

Hash statico con concatenazioni separate

Teorema

Sia $\alpha = \frac{N}{M}$. La probabilità che una data lista di una tabella

- sia lunga k è minore di

$$\frac{\alpha^k e^{-\alpha}}{k!}$$

- abbia più di $t\alpha$ dati è minore di

$$\left(\frac{\alpha e}{t}\right)^t e^{-\alpha}$$

Ad esempio, se $\alpha = 20$ la probabilità di osservare una lista con almeno 40 elementi è minore di

$$\left(\frac{20e}{2}\right)^2 e^{-20} \approx 0.0000016$$

Hash statico con indirizzamento aperto

I dati vengono inseriti direttamente in tabella. La tabella deve avere una dimensione maggiore del numero di dati attesi, $M \geq n$.
Sorge il problema di gestione delle collisioni, ovvero cosa fare se

Member(x),Delete(x): la posizione $H(x.chiave)$ contiene $y \neq x$

Insert(x): la posizione $H(x.chiave)$ è occupata

Hash statico con indirizzamento aperto

I dati vengono inseriti direttamente in tabella. La tabella deve avere una dimensione maggiore del numero di dati attesi, $M \geq n$.
Sorge il problema di gestione delle collisioni, ovvero cosa fare se

Member(x),Delete(x): la posizione $H(x.chiave)$ contiene $y \neq x$

Insert(x): la posizione $H(x.chiave)$ è occupata

Hash statico con indirizzamento aperto

I dati vengono inseriti direttamente in tabella. La tabella deve avere una dimensione maggiore del numero di dati attesi, $M \geq n$.
Sorge il problema di gestione delle collisioni, ovvero cosa fare se

Member(x),Delete(x): la posizione $H(x.chiave)$ contiene $y \neq x$

Insert(x): la posizione $H(x.chiave)$ è occupata

Hash statico con indirizzamento aperto

I dati vengono inseriti direttamente in tabella. La tabella deve avere una dimensione maggiore del numero di dati attesi, $M \geq n$.
Sorge il problema di gestione delle collisioni, ovvero cosa fare se

Member(x),Delete(x): la posizione $H(x.chiave)$ contiene $y \neq x$

Insert(x): la posizione $H(x.chiave)$ è occupata

Hash statico con indirizzamento aperto

In caso di conflitto vengono sondate altre posizioni della tabella

$$h_0(c), h_1(c), \ldots$$

dove

$$h_i(c) = \langle Hash(c) + F(i) \rangle_M$$

Le strategie si differenziano rispetto alla scelta della funzione F

Scansione linerare: $F(i) = i$

Scansione quadratica: $F(i) = i^2$

Hash doppio: $F(i) = i \cdot \mathrm{Hash}_2(c)$

Hash statico con indirizzamento aperto

In caso di conflitto vengono sondate altre posizioni della tabella

$$h_0(c), h_1(c), \ldots$$

dove

$$h_i(c) = \langle Hash(c) + F(i) \rangle_M$$

Le strategie si differenziano rispetto alla scelta della funzione F

- Scansione linerare: $F(i) = i$
- Scansione quadratica: $F(i) = i^2$
- Hash doppio: $F(i) = i \cdot \mathrm{Hash}_2(c)$

Hash statico con indirizzamento aperto

In caso di conflitto vengono sondate altre posizioni della tabella

$$h_0(c), h_1(c), \ldots$$

dove

$$h_i(c) = \langle Hash(c) + F(i) \rangle_M$$

Le strategie si differenziano rispetto alla scelta della funzione F

Scansione linerare: $F(i) = i$

Scansione quadratica: $F(i) = i^2$

Hash doppio: $F(i) = i \cdot \mathrm{Hash}_2(c)$

Hash statico con indirizzamento aperto

In caso di conflitto vengono sondate altre posizioni della tabella

$$h_0(c), h_1(c), \ldots$$

dove

$$h_i(c) = \langle Hash(c) + F(i) \rangle_M$$

Le strategie si differenziano rispetto alla scelta della funzione F

Scansione linerare: $F(i) = i$

Scansione quadratica: $F(i) = i^2$

Hash doppio: $F(i) = i \cdot \text{Hash}_2(c)$

Hash statico con indirizzamento aperto

In caso di conflitto vengono sondate altre posizioni della tabella

$$h_0(c), h_1(c), \ldots$$

dove

$$h_i(c) = \langle Hash(c) + F(i) \rangle_M$$

Le strategie si differenziano rispetto alla scelta della funzione F

Scansione linerare: $F(i) = i$

Scansione quadratica: $F(i) = i^2$

Hash doppio: $F(i) = i \cdot \mathrm{Hash}_2(c)$

Hash statico con indirizzamento aperto

Prestazioni

Il costo delle operazioni dipende da:

- fattore di carico $\alpha = \frac{N}{M}$
- dimensione dei cluster (gruppi contigui di posizioni occupate)
- gestione delle collisioni

Hash statico con indirizzamento aperto

Prestazioni

Il costo delle operazioni dipende da:

- fattore di carico $\alpha = \frac{N}{M}$
- dimensione dei cluster (gruppi contigui di posizioni occupate)
- gestione delle collisioni

Hash statico con indirizzamento aperto

Prestazioni

Il costo delle operazioni dipende da:

- fattore di carico $\alpha = \frac{N}{M}$
- dimensione dei cluster (gruppi contigui di posizioni occupate)
- gestione delle collisioni

Hash statico con indirizzamento aperto

Prestazioni

Il costo delle operazioni dipende da:

- fattore di carico $\alpha = \frac{N}{M}$
- dimensione dei cluster (gruppi contigui di posizioni occupate)
- gestione delle collisioni

Hash statico con indirizzamento aperto

Prestazioni

Nel caso di conflitti risolti tramite scansione lineare vale:

Teorema

Il numero medio di sondaggi eseguiti da una ricerca in una tabella con fattore di carico α

(successo)

$$\frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)$$

(insuccesso)

$$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$$

Hash statico con indirizzamento aperto

Prestazioni

Nel caso di conflitti risolti tramite hash doppio vale:

Teorema

Il numero medio di sondaggi eseguiti da una ricerca in una tabella con fattore di carico α

(successo)

$$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$$

(insuccesso)

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

Hash statico con indirizzamento aperto

Prestazioni

Scansione lineare

α	1/2	2/3	3/4	9/10
s	1.5	2.0	3.0	5.5
i	2.5	5.0	8.5	55.5

Hash doppio

α	1/2	2/3	3/4	9/10
s	1.4	1.6	1.8	2.6
i	1.5	2.0	3.0	5.5

Hash statico con indirizzamento aperto

Prestazioni

Scansione lineare

α	1/2	2/3	3/4	9/10
s	1.5	2.0	3.0	5.5
i	2.5	5.0	8.5	55.5

Hash doppio

α	1/2	2/3	3/4	9/10
s	1.4	1.6	1.8	2.6
i	1.5	2.0	3.0	5.5

Hash dinamico con concatenazione separate

Idea

- Non appena la tabella è piena, ($\alpha = 1$) si crea una nuova tabella di dimensione $2M$, reinserendo i dati presenti
- Quando la tabella è piena a metà, ($\alpha = 1/2$) si crea una nuova tabella di dimensione $M/2$, reinserendo i dati presenti

Hash dinamico con concatenazione separate

Idea

- Non appena la tabella è piena, ($\alpha = 1$) si crea una nuova tabella di dimensione $2M$, reinserendo i dati presenti
- Quando la tabella è piena ~~a metà~~ per un quarto, ($\alpha = 1/\cancel{2}\ 4$) si crea una nuova tabella di dimensione $M/2$, reinserendo i dati presenti

Hash dinamico con concatenazione separate

Fatto

Il costo ammortizzato di n inserimenti è $O(1)$ **(per ciascuno)**

Dimostrazione

Inseriamo i primi $M-1$ dati in una tabella di dimensione M. All'inserimento numero M creiamo una nuova tabella (dim. $2M$) e riallochiamo gli M dati. All'inserimento numero $2^i M$ passiamo da una tabella di dimensione $2^i M$ a una tabella di dimensione $2^{i+1} M$. Terminiamo quando

$$M \cdot 2^{k-1} < n \leq M \cdot 2^k$$

Costo totale

$$M + \sum_{i=0}^{\lceil \log_2(n/M) \rceil} 2^i M \leq M + M \sum_{i=0}^{\lceil \log_2 n \rceil} 2^i \leq M(1+2n) = \Theta(n)$$

Hash dinamico con concatenazione separate

Fatto

Il costo ammortizzato di n inserimenti è $O(1)$ **(per ciascuno)**

Dimostrazione

Inseriamo i primi $M-1$ dati in una tabella di dimensione M. All'inserimento numero M creiamo una nuova tabella (dim. $2M$) e riallochiamo gli M dati. All'inserimento numero $2^i M$ passiamo da una tabella di dimensione $2^i M$ a una tabella di dimensione $2^{i+1} M$. Terminiamo quando

$$M \cdot 2^{k-1} < n \leq M \cdot 2^k$$

Costo totale

$$M + \sum_{i=0}^{\lceil \log_2(n/M) \rceil} 2^i M \leq M + M \sum_{i=0}^{\lceil \log_2 n \rceil} 2^i \leq M(1+2n) = \Theta(n)$$

Hash dinamico con concatenazione separate

Fatto

Il costo ammortizzato di n inserimenti è $O(1)$ **(per ciascuno)**

Dimostrazione

Inseriamo i primi $M-1$ dati in una tabella di dimensione M. All'inserimento numero M creiamo una nuova tabella (dim. $2M$) e riallochiamo gli M dati. All'inserimento numero $2^i M$ passiamo da una tabella di dimensione $2^i M$ a una tabella di dimensione $2^{i+1} M$. Terminiamo quando

$$M \cdot 2^{k-1} < n \leq M \cdot 2^k$$

Costo totale

$$M + \sum_{i=0}^{\lceil \log_2(n/M) \rceil} 2^i M \leq M + M \sum_{i=0}^{\lceil \log_2 n \rceil} 2^i \leq M(1+2n) = \Theta(n)$$

Hash dinamico con concatenazione separate

Fatto

Il costo ammortizzato di n inserimenti è $O(1)$ **(per ciascuno)**

Dimostrazione

Inseriamo i primi $M-1$ dati in una tabella di dimensione M. All'inserimento numero M creiamo una nuova tabella (dim. $2M$) e riallochiamo gli M dati. All'inserimento numero $2^i M$ passiamo da una tabella di dimensione $2^i M$ a una tabella di dimensione $2^{i+1} M$. Terminiamo quando

$$M \cdot 2^{k-1} < n \leq M \cdot 2^k$$

Costo totale

$$M + \sum_{i=0}^{\lceil \log_2(n/M) \rceil} 2^i M \leq M + M \sum_{i=0}^{\lceil \log_2 n \rceil} 2^i \leq M(1+2n) = \Theta(n)$$

Hash dinamico con concatenazione separate

Fatto

Il costo ammortizzato di n inserimenti è $O(1)$ **(per ciascuno)**

Dimostrazione

Inseriamo i primi $M-1$ dati in una tabella di dimensione M. All'inserimento numero M creiamo una nuova tabella (dim. $2M$) e riallochiamo gli M dati. All'inserimento numero $2^i M$ passiamo da una tabella di dimensione $2^i M$ a una tabella di dimensione $2^{i+1} M$. Terminiamo quando

$$M \cdot 2^{k-1} < n \leq M \cdot 2^k$$

Costo totale

$$\cancel{M +} \sum_{i=0}^{\lceil \log_2(n/M) \rceil} 2^i M \leq \cancel{M +} M \sum_{i=0}^{\lceil \log_2 n \rceil} 2^i \leq M(\cancel{1 +} 2n) = \Theta(n)$$