

Tabelle hash

Dizionario = insieme dinamico che consente di effettuare le Operazioni di ricerca, inserimento, rimozione.

Search(S, k) - dato un insieme S ed un valore chiave k restituisce un puntatore x ad un elemento in S tale che $key[x] = k$

Insert(S, x) - inserisce in S l'elemento puntato da x

Delete(S, x) - rimuove da S l'oggetto puntato da x

Proviamo ad implementare un dizionario utilizzando un **tabella hash** → generalizzazione piu' semplice del concetto di array ordinario → *ottime prestazioni nel caso medio*

Tabelle ad indirizzamento diretto

Supponiamo che -

- ✓ Gli elementi di un insieme dinamico abbiano chiavi distinte;
- ✓ La chiave di un generico elemento dell' insieme dinamico sia contenuta in un insieme (universo) di M elementi $U = 0, 1, \dots, M-1$.
- ✓ Ad un istante generico, l'insieme dinamico contiene n , con $n \leq M$, oggetti, le cui chiavi sono contenute in U .

E' evidente che:

Per rappresentare l'insieme dinamico possiamo usare un array T di M elementi (tabella ad indirizzamento diretto), in cui ogni posizione (slot) corrisponde ad una chiave nell'universo U .

La posizione $T[k]$ della tabella contiene un puntatore all'elemento dell'insieme dinamico con chiave k , se tale elemento esiste. Se l'insieme dinamico non contiene elementi con chiave k allora $T[k] = \text{NIL}$.

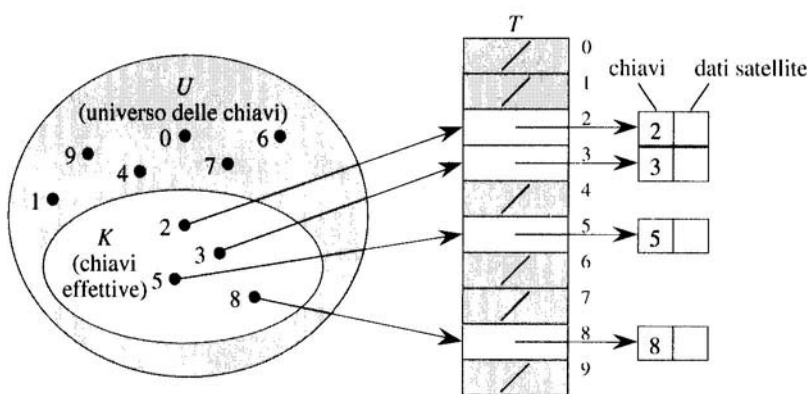


Fig. 12.1 La realizzazione di un insieme dinamico tramite una tabella ad indirizzamento diretto T . Ogni chiave nell'universo $U = \{0, 1, \dots, 9\}$ corrisponde a un indice nella tabella. L'insieme di chiavi effettive determina le posizioni nella tabella che contengono i puntatori agli elementi. Le altre posizioni, grigie scure, contengono NIL.

Tabelle ad indirizzamento diretto

Direct-Address-search(T, k)
return $T[k]$

Direct-Address-Insert(T, x)
 $T[k[x]] \leftarrow x$

Direct-Address-Delete(T, x)
 $T[k[x]] \leftarrow \text{NIL}$

Per tutte queste procedure:

$$T(n) = O(1)$$

E' evidente che:

Indirizzamento diretto = tecnica vantaggiosa quando l'universo (insieme dei possibili valori) delle chiavi e' ragionevolmente piccolo.

Tabelle hash

Se l'universo U e' molto grande la tecnica dell'indirizzamento diretto diviene impraticabile

→ $\text{lenght}(T) = \text{dim}(U) = M$

Tabelle hash

Introduciamo una funzione h detta **funzione hash** che introduce una corrispondenza tra l'universo U delle chiavi e la tabella hash $T[0,1,\dots, m-1]$:

$h(k)$ - **valore hash di k** - fornisce la posizione in T corrispondente ad un oggetto avente chiave k

→ $\text{lenght}(T) = \text{dim}(h(U)) = m < M$ (se h non biunivoca)

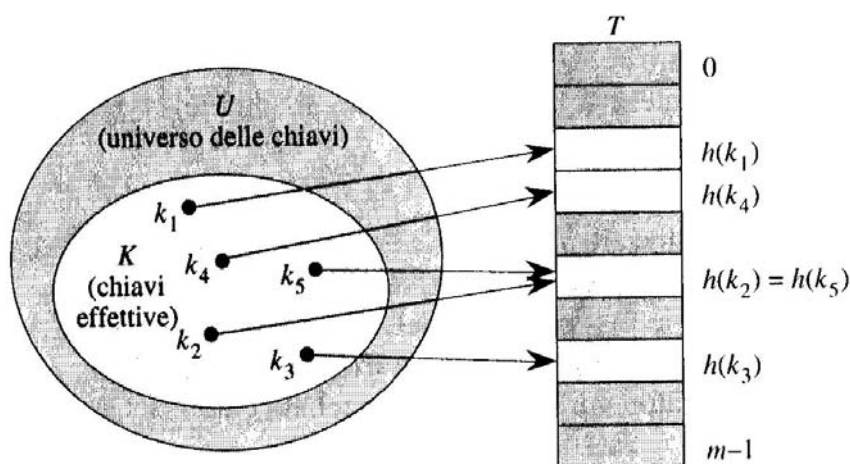


Fig. 12.2 Uso di una funzione hash h per porre in corrispondenza le chiavi e le posizioni nella tabella hash. Le chiavi k_2 e k_3 corrispondono alla stessa posizione, quindi collidono.

Problema delle collisioni - 2 o piu chiavi possono corrispondere alla stessa posizione.

N.B. - Se $\text{dim}(h(U)) < \text{dim}(U)$ → le collisioni sono inevitabili

Risoluzione delle collisioni per concatenazione

Per risolvere il problema delle collisioni si può usare la tecnica seguente:

Indichiamo con j una generica posizione della tabella hash

$T(j) \rightarrow$ contiene il puntatore alla testa di una lista contenente tutti gli elementi del dizionario con chiave k tale che $h(k) = j$.

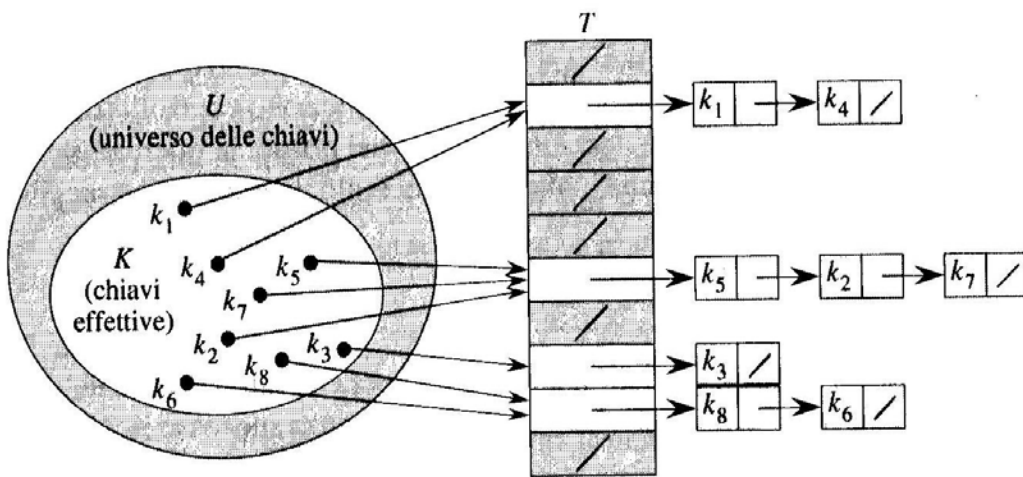


Fig. 12.3 Risoluzione di una collisione per concatenazione. Ogni posizione $T[j]$ nella tabella hash contiene una lista concatenata di tutte le chiavi il cui valore hash è j . Per esempio, $h(k_1)=h(k_4)$ e $h(k_5)=h(k_2)=h(k_7)$.

Chained-hash-insert(T, x)

$T(n) = O(1)$

Inserisci x nella testa della lista $T[h(\text{key}[x])]$

Chained-hash-delete(T, k)

$T(n) = O(1)$

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

Chained-hash-search(T, k)

$T(n) \rightarrow$ vedi dopo

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

Ricerca in una tabella hash

La ricerca di un elemento con una chiave k in una tabella hash si riduce alla ricerca nella lista concatenata $h(k)$. Il tempo di esecuzione di tale operazione dipende dal numero di elementi n_L memorizzati nella lista.

$$T(n) = O(n_L) \quad n_L = \text{num elementi nella lista}$$

Quindi in generale (il valore di chiave puo' essere qualsiasi)

$$T_{\text{hash}}(n) = O(N_L)$$

$N_L = \max(n_L)$ - lunghezza max delle liste nella tab. hash

E' evidente che N_L dipende da:

- Caratteristiche dell'input

- Se gli n oggetti hanno tutti chiavi corrispondenti alla stessa posizione $\rightarrow N_L = n \rightarrow T_{\text{hash}} = O(n)$
- Se gli n oggetti hanno chiavi corrispondenti a posizioni distinte $\rightarrow T_{\text{hash}} = O(1)$

La domanda e' allora Qual e' il tempo medio?

- Caratteristiche della funzione hash

Dato un input, diverse funzioni hash distribuiscono diversamente gli elementi tra le varie posizioni.

Analisi dell'organizzazione hash con concatenazione

Data una tabella hash T con m posizioni che memorizza n elementi, definiamo

Fattore di carico

$$\alpha = n / m$$

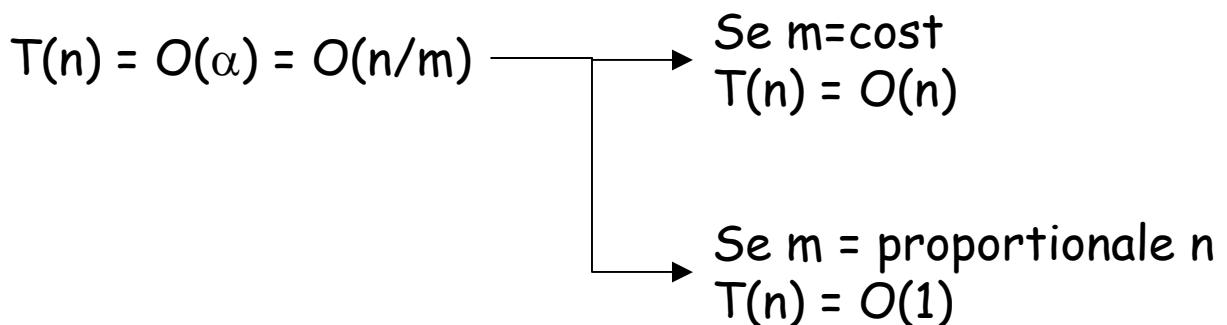
Notare - il fattore α dipende

- dalle dimensioni del dizionario
- dalle caratteristiche della funzione hash -
 $m = \dim(h(U))$

Ipotizziamo che - Ogni elemento corrisponda in modo equamente probabile ad una delle m posizioni della tabella (uniformita' semplice della funzione hash)

Sotto questa ipotesi \rightarrow Nel caso medio, tutte le liste hanno lo stesso numero di elementi.

α = numero di elementi per posizione



Funzioni hash

Requisito per una buona funzione hash

- Deve poter essere calcolata velocemente;
- Soddisfa (approssimativamente) l'ipotesi di una uniformita' semplice. A parole ...

"Ogni elemento corrisponde in modo equamente probabile ad una delle m posizioni della tabella"

In termini piu' formali ...

Supponiamo di conoscere la distribuzione di probabilita' $P(k)$ dei vari valori chiave. Richiediamo:

$$\sum_{k: h(k)=j} P(k) = \frac{1}{m}$$

→ $P(k)$ = Probabilita' che una chiave k sia estratta (in genere non nota a priori).

Un esempio ...

Il metodo di divisione

Ipotizziamo che l'universo delle chiavi sia un sotto-insieme $U = 0,1,2,\dots,M-1$ dei numeri naturali (altrimenti troviamo un modo per convertirle come tali).

Nel metodo della divisione associamo a ciascuna chiave k , la posizione j pari al resto della divisione di k per m

$$h(k) = k \bmod m \quad (\text{resto della divisione } k/m)$$

Notare - I valori possibili per $h(k)$ sono $0,1,2, \dots m-1$

Soddisfa la ipotesi di uniformita' semplice?

Ipotizziamo che i vari valori di k siano equamente probabili:

$$P(k) = 1/M$$

Abbiamo che:

$$\sum_{k: h(k)=j} P(k) = \frac{1}{M} \cdot \sum_{k: h(k)=j} 1 = \frac{\text{N. elem. corr. a } j}{M} \cong \frac{M}{m} \cdot \frac{1}{M} = \frac{1}{m}$$

Indirizzamento aperto

Nell' **indirizzamento aperto**, tutti gli elementi sono memorizzati nella tabella hash stessa (non vi sono liste ne' puntatori)

$$\alpha = n/m = \text{fattore di carico} \leq 1 \quad \rightarrow \quad n \leq m$$

Ogni elemento della tabella contiene un elemento dell'insieme dinamico o NIL.

Invece di utilizzare i puntatori, la posizione di un elemento all'interno della tabella viene **calcolata** sulla base del suo valore di chiave **k** e di un **contatore i** che conta il numero di possibili posizioni per k gia' occupate (esaminate).

La funzione hash dipende dunque da 2 variabili:

$$h(k,i) : \underset{\text{Universo}}{U} \times \underset{\text{N. nos. gia' occ.}}{0,1,\dots,m-1} \rightarrow \underset{\text{Posizione nella tabella}}{0,1,\dots,m-1}$$

Per ogni k risulta definita una **sequenza di scansione**:

$$\langle h(k,0), h(k,1), \dots, h(k,m-1) \rangle = \text{permutazione di } \langle 0,1,\dots,m-1 \rangle$$

Vantaggi dell'indirizzamento aperto - La memoria liberata dai puntatori puo' essere utilizzata per memorizzare nuovi elementi (riducendo la possibilita' di collisioni).

Indirizzamento aperto

Hash-Insert(T, k)

$i \leftarrow 0$

```
Repeat   $j \leftarrow h(k, i)$ 
        if ( $T[j] = \text{NIL}$ )
            then  $T[j] \leftarrow k$ 
                return  $j$ 
        else  $i \leftarrow i + 1$ 
```

Until $i \leftarrow m$

Error "overflow sulla tabella hash"

Hash-Search(T, k)

$i \leftarrow 0$

```
Repeat   $j \leftarrow h(k, i)$ 
        if ( $T[j] = k$ )
            then return  $j$ 
         $i \leftarrow i + 1$ 
```

Until ($T[j] = \text{NIL}$) o ($i = m$)
return NIL

Cancellazione - complessa perche' non abbiamo puntatori al successivo

Analisi della complessita'

La complessita' delle varie procedure dipende da quanto "a fondo" dobbiamo effettuare la scansione della tabella:

Caso peggiore

$T(n) = O(m)$ scansione di tutta la tabella

Caso medio

Ipotizziamo che la funzione $h(k,i)$ "mappi" in maniera uniforme l'Universo U nella tabella hash T (uniformita' della funzione di hash). Cioe':

Ogni elemento dell'insieme dinamico, *ad ogni istante*, corrisponde in modo equamente probabile ad una delle posizioni della tabella:

N.B. Abbiamo un "dimensione" in piu' rispetto al caso precedente (la variabile i tiene conto della storia passata).
→ Generalizzazione del concetto di uniformita' semplice.

Ci aspettiamo che, ad ogni "estrazione" una posizione j abbia probabilita' di essere occupata pari a $P_{occ} = 1/m$. Dunque, dopo n estrazioni la probabilita' di avere avuto collisioni (probabilita' di avere provato ad occupare un elemento j piu' di una volta) e' $P_{coll} = n / m$.

Il numero di collisioni *nel caso medio* e' dunque pari a $n/m = \alpha < 1$

→ $T(n) = O(1)$