STRUTTURE DATI
-DIZIONARI-

Dizionari

Struttura dati di forma (chiave, elemento) dove:

- Chiave = meccanismo di accesso all'elemento ed è univoco
- Elemento = è un qualsiasi tipo

Operazioni = ricerca, inserimento, cancellazione

dizionario ordinato = elementi ordinati rispetto alle chiavi

dizionario non ordinato = elementi non ordinati

	Insert	Find	Delete
Lista non ordinata	O(1)	O(n)	O(n)
Alberi	O(log n)	O(log n)	O(log n)
Array ordinato	O(n)	O(log n)	O(n)
ind. diretto	O(1)	O(1)	O(1)

dati studente	(matricola, dati studente)
vocabolario	(parola, definizione)
tabella dei simboli	(nome della variabile, indirizzo della variabile)

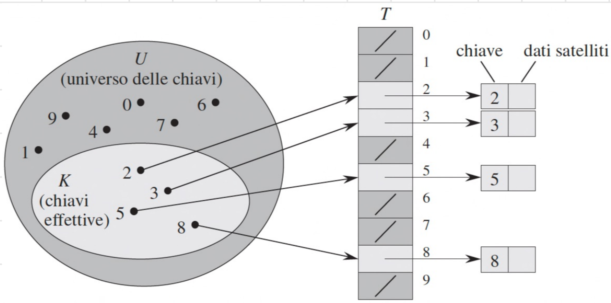
Tavole a indirizzamento diretto

Tecnica efficace quando non dobbiamo memorizzare tante chiavi $U=\{0,1,\dots,m-1\}$ per rappresentare l'insieme dinamico utilizzo un array $T[0,\dots,m-1]$ dove ogni posizione corrisponde ad un indice preso da U, se l'insieme non contiene la chiave k allora $T[k]=nil$

```
DIRECT-ADDRESS-SEARCH(T, k)
  return T[k]

DIRECT-ADDRESS-INSERT(T, x)
  T[x.key] = x

DIRECT-ADDRESS-DELETE(T, x)
  T[x.key] = NIL
```



Un elemento con chiave k è memorizzato nella cella $T[k]$

Tavole hash

L'hashing utilizza una funzione hash per calcolare la cella della chiave K $h: U \to \{0, \dots, m-1\}$

La dimensione m della tavola hash è generalmente più piccola di |U|

chiave K → h(K)
elemento — cella

h(K) è anche il valore hash della chiave K

Quando la quantità di chiavi da memorizzare è minore di U allora una tabella hash è più efficace rispetto tavola a indirizzamento diretto

Spazio richiesto: $\Theta(|K|)$ con tempo di ricerca $O(1)$

Funzione hash = riduce l'intervallo degli indici e la dimensione dell'array

Probing = Scandisco la tabella finché non trovo una posizione libera secondo un criterio (lineare, quadratico, doppio hash)

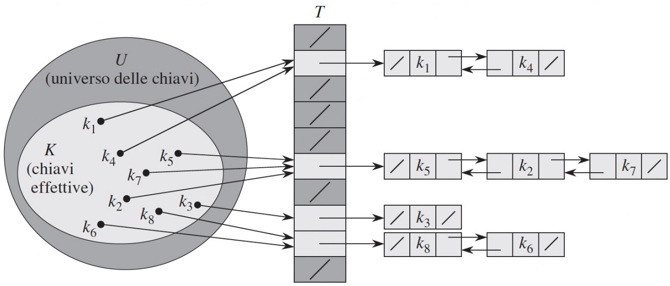
Collisione

Due chiavi mappate nella stessa cella, esistono diverse tecniche per evitare ciò

Concatenamento

Tecnica più semplice. Poniamo tutti gli elementi con chiavi in comune in una lista concatenata

Le seguenti liste possono essere singolarmente o doppiamente concatenate, così da avere delle operazioni più efficaci



CHAINED-HASH-INSERT(T, x)
inserisce x in testa alla lista $T[h(x.key)]$

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 $T[h(x.key)]$

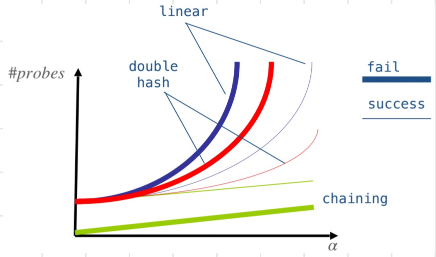
	Insert	Find	Delete
Lista non ordinata	O(1)	O(N)	O(N)
Alberi	O(log N)	O(log N)	O(log N)
Array ordinato	O(N)	O(log N)	O(N)
indirizzamento diretto	O(1)	O(1)	O(1)
hash (chaining) pessimo	O(1)	O(N)	O(N)
hash (chaining) medio	$\Theta(1)$	$\Theta(1+\alpha)$	$\Theta(1+\alpha)$
hash open addressing medio	$O(1/1-\alpha)$	$O(1/1-\alpha)$ fail $O((1/\alpha)ln(1/1-\alpha))$ success	$O(1/1-\alpha)$
hash open addressing pessimo	O(N)	O(N)	O(N)

Analisi dell'hashing con concatenamento

- Fattore di carico: $\alpha = n/m$ (n: elementi $\wedge$ m: celle)

Caso peggiore: tutte le chiavi sono associate alla stessa cella creando una lista di lunghezza n

Caso medio: dipende da come la funzione hash distribuisce l'insieme delle chiavi da memorizzare in m celle



Ricerca senza successo: $\Theta(1+\alpha)$ nel caso medio

Ricerca con successo: $\Theta(1+\alpha)$ nel caso medio

Funzioni hash

Una buona funzione hash soddisfa l'ipotesi dell'hashing uniforme semplice, ogni chiave ha la stessa probabilità di essere mandata in una qualsiasi delle m celle, ma di solito non è possibile verificare questa condizione

A volte la distribuzione è nota: $\forall k \in \mathbb{R} \wedge 0 \leq k < 1 \Rightarrow h(k) = \lfloor km \rfloor$ è un hashing semplice

il metodo della divisione

una chiave k viene associata a una delle m celle prendendo il resto della divisione fra k e m

$h(k) = k \bmod m$

m(celle) non deve mai essere una potenza di 2

il metodo della moltiplicazione

si svolge in due passi:

- moltiplichiamo la chiave k per una costante A nell'intervallo $0 < A < 1$ ed estraiamo la parte frazionaria di kA
- moltiplichiamo questo valore per m e prendiamo la parte inferiore del risultato

$$h(k) = \lfloor m \underbrace{(kA \bmod 1)}_{=} \rfloor$$

$\rightarrow kA - \lfloor kA \rfloor$ ovvero la parte frazionaria di kA

il valore di m non è critico, tipicamente è una potenza di 2 ($m = 2^p$ per qualche intero p)

Hash universale

Scegliere casualmente la funzione hash in modo che sia indipendente dalle chiavi che devono essere memorizzate
All'inizio dell'esecuzione dell'algoritmo, viene scelta casualmente la funzione in una classe di funzioni pre progettate

Queste modalità garantiscono buone prestazioni nel caso medio con qualsiasi input

Con hashing universale e la risoluzione delle collisioni mediante concatenamento, occorre $\Theta(n)$ per eseguire:

- Insert
- Search
- Delete

poiché il numero di inserimenti è $O(m)$, si ha $n = O(m)$, quindi $\alpha = O(1)$

Progettare una classe universale di funzioni hash

Supponiamo che la dimensione dell'universo delle chiavi sia maggiore del numero di celle nella tavola hash

$p > m \quad \forall a \in \mathbb{Z}_p^* \land \forall b \in \mathbb{Z}_p$

- $h_{ab}(k) = ((ak + b) \bmod p) \bmod m$

Famiglia di tutte le funzioni hash: $\mathcal{H}_{pm} = \{ h_{ab} : a \in \mathbb{Z}_p^* \land b \in \mathbb{Z}_p$

Indirizzamento aperto

tutti gli elementi sono memorizzati nella tavola hash, non ci sono liste né elementi memorizzati all'esterno della tavola.
Può riempirsi al punto tale che non possono essere effettuati altri inserimenti. (α non supera mai 1)

- possiamo inserire liste per il concatenamento all'interno della tavola hash

```
HASH-INSERT(T, k)
i = 0
repeat
    j = h(k, i)
    if T[j] == NIL
        T[j] = k
        return j
    else i = i + 1
until i == m
error "overflow della tavola hash"
```

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

- Hash-Insert = restituisce il numero della cella in cui ha memorizzato la chiave K
- Hash-search = restituisce j se nella cella j è contenuta la chiave K

Ispezione lineare

Applica una funzione hash su tutte le celle m della tabella hash. La prima cella ispezionata determina l'intera sequenza di ispezioni.

Ci sono m sequenze di ispezione distinte

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

Addensamento primario

Si formano lunghe file di celle occupate, che aumentano il tempo medio di ricerca.

- Una cella vuota preceduta da i celle piene ha la probabilità $(i+1)/m$ di essere la prossima a essere occupata

le lunghe file hanno una crescita esponenziale

Ispezione quadratica

Usa la funzione hash della forma: $h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$

- h': funzione ausiliaria
- $c_1 \wedge c_2 \neq 0$: costanti ausiliarie

per funzionare al meglio, c_1, c_2, m non si possono scegliere

Addensamento secondario

se due chiavi hanno la stessa posizione iniziale di ispezione, allora le loro sequenze di ispezione sono identiche

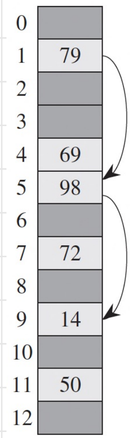
perché: $h(k_1, 0) = h(k_2, 0) \Rightarrow h(k_1, i) = h(k_2, i)$

Doppio hashing

È uno dei metodi migliori disponibili per l'indirizzamento aperto, perché le permutazioni prodotte hanno molte delle permutazioni scelte a caso

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

Funzioni hash ausiliarie



Inizia dalla posizione $T[h_1(k)]$ e scorre di $h_2(k) \bmod m$ posizioni

- Può variare sia la posizione iniziale di ispezione sia la distanza fra due posizioni successive di ispezione

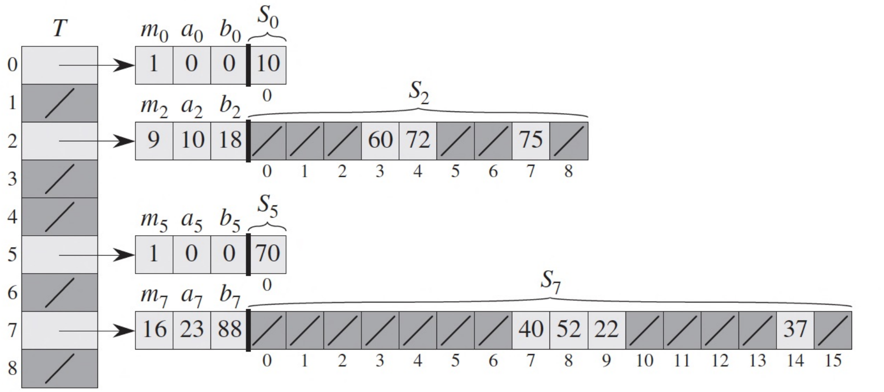
- Per garantire l'ispezione dell'intera tavola hash è una buona pratica scegliere m potenza di 2 e definire h_2 in modo che produca sempre un numero dispari, oppure m primo e definire h_2 in modo che produca sempre un numero intero positivo minore di m

Hashing perfetto

Tecnica hashing secondo la quale il numero di accessi in memoria richiesti per svolgere una ricerca è $O(1)$ nel caso peggiore.
Per crearne uno, utilizziamo lo schema di hashing a due livelli, con un hashing universale per ogni livello.

- il primo livello è l'hashing con concatenamento ma al posto di una lista delle chiavi associate alla cella j si utilizza una piccola tavola hash secondaria S_j con una funzione h_j associata.

Scegliendo le funzioni hash accurate evitiamo le collisioni al livello secondario, ma per garantire ciò, la dimensione m della tavola hash secondaria S_j dovrà il quadrato del numero n_j delle chiavi che si associano alla cella j



Se memorizziamo n chiavi in una tavola hash di dimensione $m = n^2$ utilizzando una funzione h avremo $\binom{n}{2}$ coppie di chiavi che potremmo collidere con un probabilità $1/m$ se h è scelta a caso

Se memorizziamo n chiavi in una tavola hash di dimensione $m = n$ utilizzando una funzione scelta a caso da una classe universale di funzioni hash, si ha:

$$E\left[\sum_{j=0}^{m-1} n_j^2\right] < 2n$$ dove n_j è il numero delle chiavi mandate nella cella j