

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica



Corso di Algoritmi e Strutture Dati

Tabelle hash





DIE
TI.
UNI
NA



DIE
TI.
UNI
NA

Tabelle a indirizzamento diretto

- Le liste concatenate possono essere utilizzate per implementare dizionari, ovvero insiemi dinamici che supportano le operazioni di inserimento, eliminazione e ricerca di un elemento
- Le tavole a indirizzamento diretto sono utilizzabili quando le chiavi appartengono all'insieme $\{0, 1, \dots, m-1\}$ con m non troppo grande
- Si usa un array con m posizioni
 - Nella posizione k si memorizza l'elemento la cui chiave è k
 - Gli elementi devono avere chiavi distinte
 - Se l'elemento con chiave k non è presente nell'insieme, nella posizione k si memorizza il puntatore NIL

Tabelle a indirizzamento diretto



Direct-Address-Search (T, k)

return $T[k]$

Direct-Address-Insert (T, x)

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

Direct-Address-Delete (T, x)

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

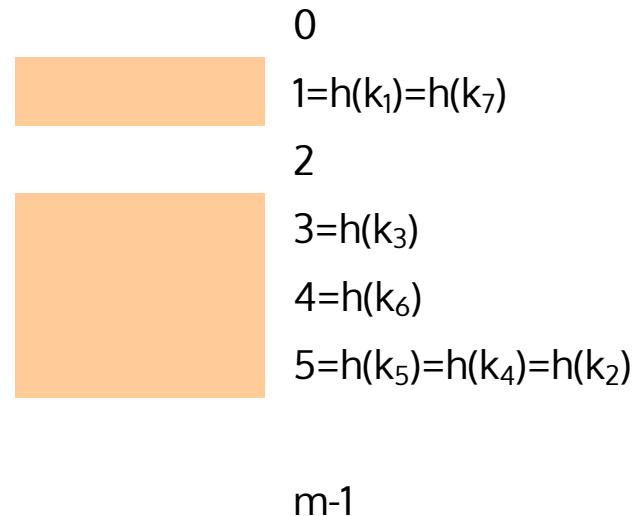
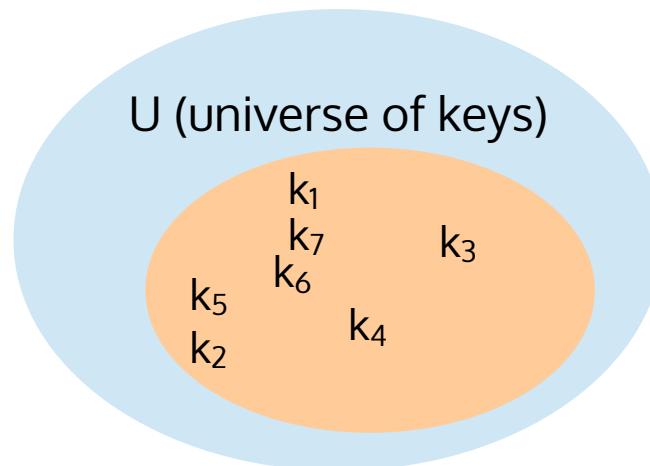
Tutte le operazioni si realizzano in un tempo $O(1)$

Tabelle hash

- Se la cardinalità dell'insieme delle possibili chiavi è molto elevata, può essere difficile (se non impossibile) utilizzare le tabelle a indirizzamento diretto
 - E si spreca una grossa quantità di memoria se le chiavi effettivamente utilizzate sono poche
- Quando il numero di chiavi da memorizzare è molto minore della cardinalità dell'insieme delle possibili chiavi U , le tabelle hash costituiscono una efficiente soluzione
- La quantità di memoria richiesta è proporzionale al numero di chiavi da memorizzare
- Accedere ad un elemento richiede $O(1)$ *nel caso medio*

Tabelle hash

- In una tabella hash, un elemento con chiave k viene memorizzato in posizione $h(k)$, dove h è una funzione hash
- $h: U \rightarrow \{0, 1, \dots, m-1\}$



- Quando la funzione hash applicata a più chiavi restituisce lo stesso valore si produce una *collisione*

Tabelle hash



DIE
TI.
UNI
NA

- Non è possibile evitare le collisioni
 - La funzione hash si sceglie in modo da minimizzarne la probabilità
- La tecnica della concatenazione (**chaining**) prevede di memorizzare tutti gli elementi che collidono in una lista concatenata

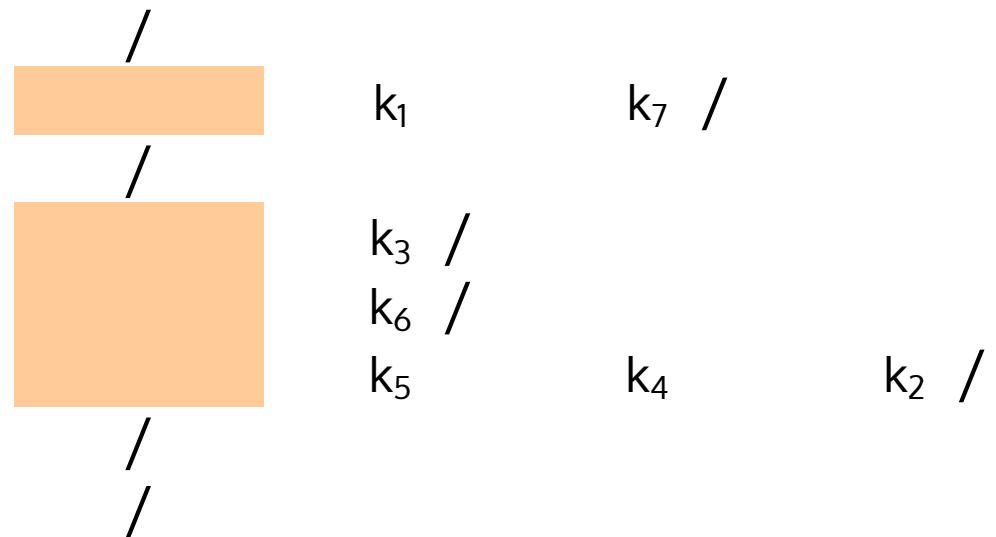
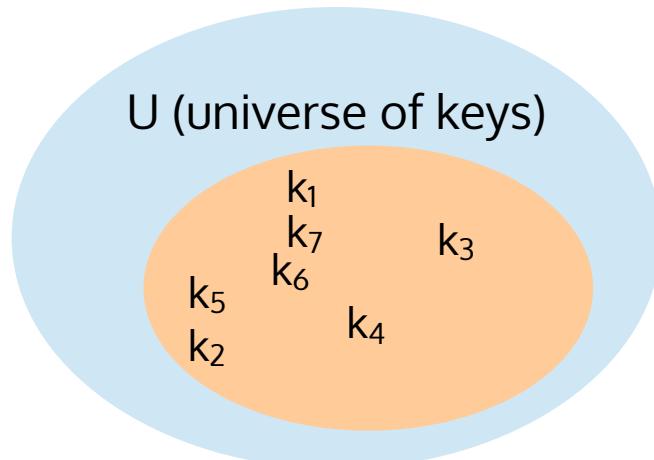


Tabelle hash

Chained-Hash-Insert (T, x)

insert x at the head of list $T[h(\text{key}[x])]$

Chained-Hash-Search (T, k)

search for an element with key k in list $T[h(k)]$

Chained-Hash-Delete (T, x)

delete x from the list $T[h(\text{key}[x])]$

Poichè l'insert è in testa alla lista.

- L'inserimento richiede un tempo $O(1)$ nel caso peggiore
- L'eliminazione (dato l'elemento, non la chiave) richiede un tempo $O(1)$ se le liste sono doppiamente concatenate
- Vediamo il tempo richiesto per la ricerca

Tabelle hash

- Nel caso peggiore, la ricerca di un elemento richiede $\Theta(n)$
 - Tutti gli n elementi sono inseriti nella stessa lista concatenata
- Le prestazioni di una tabella hash dipendono dalla capacità della funzione hash di distribuire gli elementi tra le liste concatenate
- Supponiamo che ogni elemento abbia la stessa probabilità di finire in ciascuna delle liste concatenate, indipendentemente da dove sono finiti gli altri elementi (*simple uniform hashing*)
- Detta n_j la lunghezza della j -esima lista
- $n = n_0 + n_1 + \dots + n_{m-1}$
- $E[n_j] = n/m = \alpha$

Tabelle hash



DIE
TI.
UNI
NA

- Calcoliamo il numero atteso di elementi esaminati dalla funzione search
 - **Caso 1:** la ricerca di un elemento con chiave k non ha successo
 - Una chiave non presente nella tabella ha una uguale probabilità di essere associata ad una delle m liste
 - Il tempo atteso in questo caso è il tempo per cercare fino alla fine della lista $h(k)$, che ha lunghezza media α
 - Il tempo impiegato dalla ricerca è dunque $\Theta(1+\alpha)$
Considerando anche il tempo per il calcolo di $h(k)$
- +1 perchè si considerà il tempo costante di accesso al hash map

Tabelle hash

. Caso 2: la ricerca di un elemento x con chiave k ha successo

- Il numero di elementi esaminati durante la ricerca è 1 più il numero di elementi che precedono x nella sua lista
- Tali elementi sono stati inseriti successivamente a x
- Mediamo su tutti gli elementi x' in tabella, la quantità 1 più il numero di elementi inseriti successivamente a x' nella lista di x'
- Sia x_i l' i -esimo elemento inserito in tabella, $k_i = \text{key}[x_i]$,
 $X_{ij} = I[h(k_i) = h(k_j)]$
- $\Pr\{h(k_i) = h(k_j)\} = 1/m \Rightarrow E[X_{ij}] = 1/m$

n perchè si condirano gli elementi inseriti dopo

$$\begin{aligned} E \left[\frac{1}{n} \sum_{i=1}^n \left(1 + \sum_{j=i+1}^n X_{ij} \right) \right] &= \frac{1}{n} \sum_{i=1}^n \left(1 + \sum_{j=i+1}^n E[X_{ij}] \right) = \frac{1}{n} \sum_{i=1}^n \left(1 + \sum_{j=i+1}^n \frac{1}{m} \right) = 1 + \frac{1}{nm} \sum_{i=1}^n (n - i) \\ &= 1 + \frac{1}{nm} \left(\sum_{i=1}^n n - \sum_{i=1}^n i \right) = 1 + \frac{1}{nm} \left(n^2 - \frac{n(n+1)}{2} \right) = 1 + \frac{n-1}{2m} = 1 + \frac{\alpha}{2} - \frac{\alpha}{2n} = \Theta(1 + \alpha) \end{aligned}$$

Tabelle hash

- Se il numero di liste m è almeno proporzionale al numero di elementi nella tabella n , abbiamo $n = O(m)$
- Quindi $\alpha = n/m = O(m)/m = O(1)$
- *La ricerca mediamente richiede un tempo costante, se vale la proprietà del simple uniform hashing* 

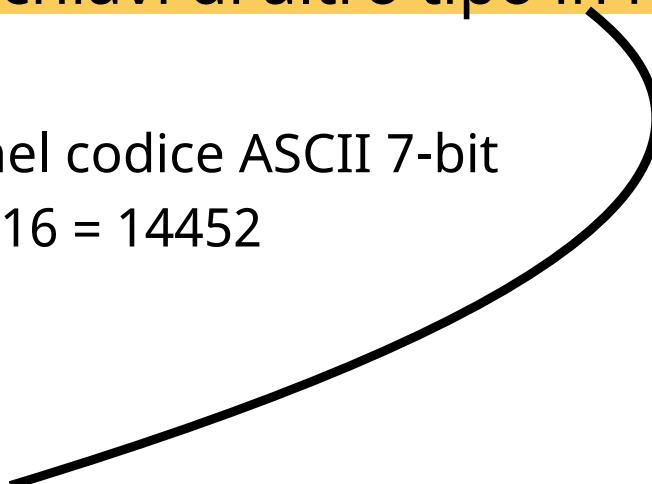
Le liste hanno più o meno tutte la stessa grandezza

- Tale proprietà è difficile da garantire, specialmente se non è nota la distribuzione di probabilità delle chiavi
- Si cerca comunque di definire delle funzioni hash che abbiano buone prestazioni anche quando tale distribuzione non è nota

Insomma avrei bisogno di una funzione hash basato sul conoscere la pdf di una k che in questo caso è una mia v.a.

Funzioni hash

- Le funzioni tipicamente assumono che le chiavi appartengano all'insieme dei numeri naturali
- È facile convertire chiavi di altro tipo in numeri naturali
 - es. stringa "pt"
 - 'p'=112 e 't'=116 nel codice ASCII 7-bit
 - "pt" → $112 \times 128 + 116 = 14452$



Questa cosa è detta PRE-hashing

Il metodo della divisione



DIE
TI.
UNI
NA

- La funzione hash è $h(k) = k \bmod m$
 - Operazione abbastanza veloce, restituisce valori in $[0..m-1]$
- È opportuno evitare alcuni valori per m
- Se $m = 2^p$, $h(k)$ è dato dagli ultimi p bit della rappresentazione di k
 - È meglio che la funzione hash dipenda dal valori di tutti i bit
- In genere, una buona scelta è un numero primo non troppo vicino ad una potenza di 2
- Se abbiamo una stima di n e tolleriamo liste di, ad esempio, 3 elementi, possiamo scegliere m come numero primo intorno a $n/3$ *Valore a cazzo*

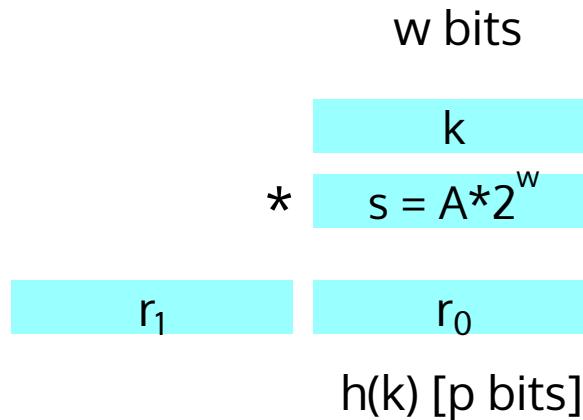
Il metodo della moltiplicazione



DIE
TI.
UNI
NA

L'obiettivo è che la mia h dipenda dai bit centrali.

- La funzione hash è $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$
 - Moltiplica k per una costante $0 < A < 1$ e prende la parte decimale
 - Moltiplica il risultato per m e lo approssima per difetto
- Implementazione efficiente con $m=2^p$ e $A=s/2^w$ con s intero $< 2^w$
 - w è il numero di bit di una *word* (sufficienti per rappresentare una chiave)



Esempio

$w=4$ bits, $k=6$, $p=3$, $s=9 \Rightarrow A=9/16=0,5625$	
0110	
* 1001	
110110	
$k*s$	
$k*A=(k*s)/2^w$	11,0110
Parte decimale	0,0110
Molt. per $m=2^p$	11,0
Parte intera	11 (=3)

Hashing universale



DIE
TI.
UNI
NA

Soluzione migliore

- Se la funzione di hash è nota, si può determinare un insieme di chiavi che vengono associate tutte alla stessa lista
- L'unica contromisura è rendere la funzione hash *aleatoria*
- L'idea dell'hashing universale è quella di scegliere la funzione di hash (all'inizio dell'esecuzione) in maniera casuale da una classe di funzioni appositamente progettata
- Nessuna sequenza di chiavi può ricadere sempre nel caso peggiore Cioè ci sono al massimo due fun hash che per valori diversi causano collisione
- Sia H una classe finita di funzioni hash che mappano un dato universo di chiavi U sull'insieme $\{0, 1, \dots, m-1\}$. Tale classe è detta universale se per ogni coppia di chiavi distinte $k, l \in U$, il numero di funzioni hash $h \in H$ per cui $h(k)=h(l)$ è al più $|H|/m$
 - La possibilità di collisione è al più $1/m$

Discorso simile al randomize quick-sort in quanto sto rendendo aleatorio il pivot, diminuendo la probabilità che per ogni passo di divide et impera trovo vettori già ordinati->il caso peggiore!

Hashing universale

- Un modo per progettare una classe universale di funzioni hash
- Sia p un numero primo sufficientemente grande da assicurare che ogni chiave k è nell'intervallo $0..p-1$
- Consideriamo $Z_p = \{0, 1, 2, \dots, p-1\}$ e $Z_p^* = \{1, 2, \dots, p-1\}$ Uguali tranne che per lo 0
- $h_{a,b}(k) = ((ak+b) \text{ mod } p) \text{ mod } m$ con $a \in Z_p^*$, $b \in Z_p$
 - Ci sono $p(p-1)$ funzioni in questa classe
 - m non deve necessariamente essere un numero primo
- Si può dimostrare che vale la proprietà dell'hashing universale

a E b presi a caso

La dimostrazione che questa è universale è nelle note

Indirizzamento aperto

- Non si usano liste concatenate, quindi tutti gli elementi sono inseriti nell'array
- Le collisioni si risolvono consentendo ad un elemento con una data chiave di trovarsi in un insieme di possibili posizioni
- In fase di inserimento e ricerca, si valutano solo tali posizioni
- Formalmente, estendiamo la funzione hash in modo da essere funzione sia della chiave che del numero di "tentativi"
 - $h : U \times \{0,1,\dots,m-1\} \rightarrow \{0,1,\dots,m-1\}$ Sequenza di ispezione
 - È richiesto che la sequenza di *probing* $\langle h(k,0), h(k,1), \dots, h(k,m-1) \rangle$ sia una permutazione di $\langle 0,1,\dots,m-1 \rangle$ per ogni chiave k
 - Consente di riempire completamente l'array

Linear probing



DIE
TI.
UNI
NA

- Data la funzione hash *ausiliaria* $h' : U \rightarrow \{0, 1, \dots, m-1\}$ considera
- $h(k, i) = (h'(k) + i) \bmod m$ con $i=0, 1, \dots, m-1$
- Le posizioni che vengono provate sono
- $h'(k), h'(k)+1, h'(k)+2, \dots, m-1, 0, 1, 2, \dots, h'(k)-1$

0		
1	79	$h'(k) = k \bmod 13$
2		
3		
4	69	$k=43 \Rightarrow h'(43) = 43 \bmod 13 = 4$
5	98	
43		
6		$(4+0) \bmod 13 = 4$ occupato
7	72	$(4+1) \bmod 13 = 5$ occupato
8		$(4+2) \bmod 13 = 6$ libero
9		
10		
11	50	
12		

Quadratic probing



- Data la funzione hash *ausiliaria* $h' : U \rightarrow \{0,1,\dots,m-1\}$ considera
- $h(k,i) = (h'(k) + c_1i + c_2i^2) \text{ mod } m$ con $i=0, 1, \dots, m-1$
- La prima posizione provata è $h'(k)$, le successive sono separate di un offset che cresce con legge quadratica rispetto ad i
- I valori di c_1 , c_2 e m sono vincolati
- Se $h(k_1,0)=h(k_2,0)$ allora $h(k_1,i)=h(k_2,i)$ per ogni i
 - Come per il linear probing, il valore provato all'inizio determina l'intera sequenza di tentativi
 - Ci sono solo m distinte sequenze di tentativi

Double hashing



DIE
TI.
UNI
NA

- Date le funzioni hash *ausiliarie* $h_1, h_2 : U \rightarrow \{0, 1, \dots, m-1\}$ usa
- $h(k, i) = (h_1(k) + i * h_2(k)) \bmod m$ con $i=0, 1, \dots, m-1$
- La prima posizione provata è $h_1(k)$, le successive sono separate di un offset che dipende anch'esso da k (tramite $h_2()$)
Mod di h_2 deve essere primo

0	
1	79
2	
3	
4	69
5	98
6	
7	72
8	
9	
10	
11	50
12	

$$h_1(k) = k \bmod 13$$
$$h_2(k) = 1 + (k \bmod 11)$$

$$k=14 \Rightarrow h_1(14) = 14 \bmod 13 = 1$$
$$h_2(14) = 1 + (14 \bmod 11) = 4$$

$$(1+0*4) \bmod 13 = 1 \text{ occupato}$$
$$(1+1*4) \bmod 13 = 5 \text{ occupato}$$
$$(1+2*4) \bmod 13 = 9 \text{ libero}$$

Double hashing

- $h_2(k)$ non deve avere fattori in comune con m
 - es. $h_1(k)=7$, $h_2(k)=4$, $m=10$, le posizioni provate sono
 - 7, 1, 5, 9, 3, 7, 1, 5, 9, 3, 7, ...
- Possibili modi per garantire questa condizione
 - m è una potenza di 2 e h_2 restituisce sempre un numero dispari
 - m numero primo e h_2 restituisce sempre un intero minore di m :
 - $h_1(k)=k \bmod m$, $h_2(k)=1+(k \bmod m')$ con $m' < m$ (es. $m-1$)
- Le prestazioni del double hashing sono generalmente buone

Il numero di sequenze di tentativi diversi è proporzionale al quadrato di m

altrimenti non avrei tutte le permutazioni possibili della sequenza di ispezione -> non sfrutto tutta la memoria disponibile

Inserimento di un elemento



- Per inserire un elemento si provano tutte le posizioni determinate dalla sequenza dei tentativi fino a che non si trova una libera

```
Hash-Insert (T,k)
    i < 0
    repeat           Ispezione lineare
        j < h(k,i)
        if T[j] = NIL
            then T[j] <- k
            return j
        else i < i+1
    until i = m
    error "hash table overflow"
```

Ricerca di un elemento

- Per cercare un elemento si provano tutte le posizioni determinate dalla sequenza dei tentativi fino a che non si trova una libera
 - In tal caso, l'elemento non è presente in tabella

```
Hash-Search (T,k)
    i < 0
    repeat
        j < h(k,i)           Ispezione lineare
        if T[j] = k
            then return j
        i < i+1
    until T[j] = NIL or i = m
    return NIL
```

Eliminazione di un elemento

- Se eliminiamo una chiave, non possiamo semplicemente sostituirla con il valore NIL
 - Non funzionerebbe l'algoritmo di ricerca per gli elementi inseriti successivamente in altre posizioni perché la posizione dell'elemento eliminato era occupata
- Possiamo utilizzare un altro valore speciale DELETED

Hash-Insert (T,k)

```
i < 0
repeat
j < h(k,i)
if T[j]=NIL or T[j]=DELETED
  then T[j] <- k
    return j
  else i < i+1
until i = m
error "hash table overflow"
```

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
```



Eliminazione di un elemento

- Quando utilizziamo il valore DELETED, però, il tempo di ricerca non dipende più dal carico $\alpha=n/m$
- Quando è necessario consentire l'eliminazione delle chiavi, le tabelle hash con liste concatenate sono da preferire

poichè gli elementi cancellati non sono considerati nel fattore n
ma sono comunque controllati nella ricerca e nel inserimento!

Analisi dell'indirizzamento aperto



- Dato che $n \leq m$, il carico $\alpha \leq 1$
- Assumiamo l'ipotesi che tutte le $m!$ permutazioni di $<0, 1, \dots, m-1>$ hanno la stessa probabilità di essere una sequenza di tentativi (*uniform hashing*)
 - In realtà, linear e quadratic probing generano solo m permutazioni e double hashing ne genera m^2
- Il numero di tentativi da effettuare prima che la ricerca si conclude con un insuccesso è $1/(1-\alpha) = 1 + \alpha + \alpha^2 + \alpha^3 + \dots$
- α è la probabilità di trovare una posizione occupata (nell'ipotesi di uniform hashing)
- Il numero di tentativi da effettuare prima che la ricerca si conclude con un successo è $1/\alpha * \ln 1/(1-\alpha)$

0 < Alpha < 1

No dimostrazioni ! O yea

#tentativi => 1/p -> #tentativi => 1/(1-alpha)

Analisi dell'indirizzamento aperto



- Numero di tentativi
 - Non dipende da n o m singolarmente, ma dal loro rapporto

α	0,5	0,9
Insuccesso	2	10
Successo	<1,4	<2,6

Nella pratica $\alpha < 0.7$