Queste slide sono un riadattamento delle slide sull'argomento del corso di Algoritmi e Strutture Dati

- Hashing
  - Tabelle hash
  - Funzioni hash e metodi per generarle
  - Inserimento e risoluzione delle collisioni
  - Eliminazione

Richiamo sul concetto di dizionario

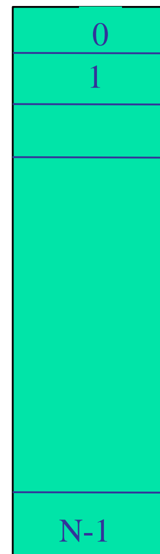
- Insieme di coppie del tipo <elemento, chiave>
- Operazioni:
  - insert(el, key)
  - delete(key)
  - search(key)

Indirizzamento diretto

- Indirizzamento diretto: si associa ad ogni valore della chiave un indice di un array – ricerca in tempo $O(1)$
- Problemi?

Indirizzamento diretto

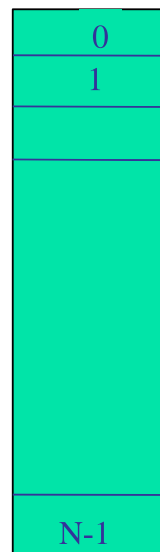
- Ogni possibile chiave corrisponde a el. diverso dell'array
- Può portare a spreco di memoria
- Es.: 10000 studenti e matr.= No. decimale a 5 cifre



No. chiavi

Obiettivi

- N ~ No. Chiavi effettivamente usate
- Tempo di ricerca $O(1)$
- D.: possibile?
- Nota: No. Chiavi possibili può essere >> N



No. chiavi

Tabella hash

Dato l'insieme base di un dizionario:

- <T, h>
- T è una tabella con N celle
- h: K → {0,…,N-1}
  - K insieme delle possibili chiavi
  - {0,…,N-1} insieme delle posizioni nella tabella, dette *pseudochiavi*

Funzioni hash perfette e collisioni

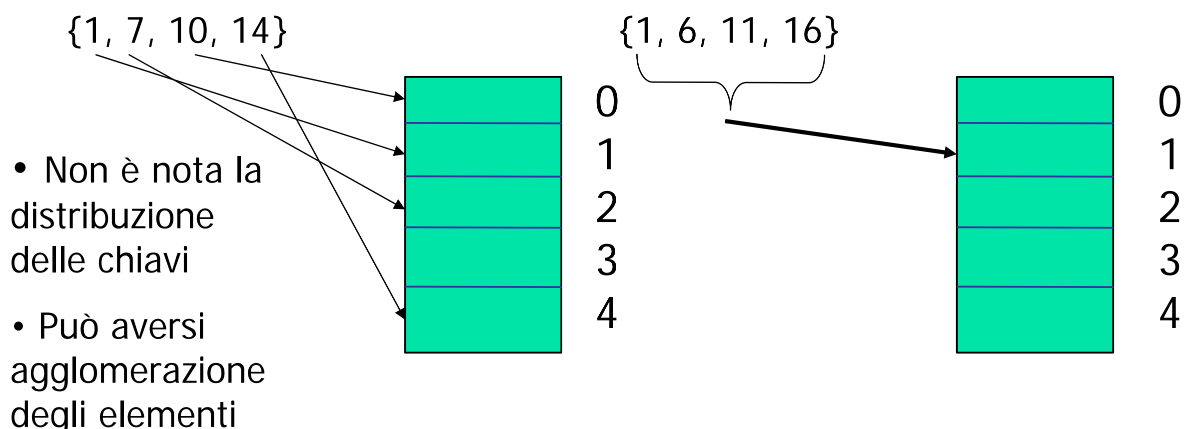
- Funzione hash perfetta:
  - $k_1 != k_2 \rightarrow h(k_1) != h(k_2)$
  - Richiede N >= |K|
  - Raramente ragionevole in pratica
- In generale N < |K| (spesso N << |K|)
  - Conseguenza: $k_1 != k_2$ ma $h(k_1) == h(k_2)$ è possibile → Collisione
  - Paradosso del Compleanno
- Es.: proporre una funzione hash perfetta nel caso in cui le chiavi siano stringhe di lunghezza 3 sull'alfabeto {a, b, c}

Requisiti di una funzione hash

- Uniformità semplice: Pr[h(k)=j] ~ 1/|K|
  - La probabilità è calcolata rispetto alla distribuzione delle chiavi
  - Intuitivamente, si desidera che gli elementi si distribuiscano nell'array in modo uniforme
  - Difficile costruire funzioni che soddisfino la proprietà
  - D.: perché?

Requisiti di una funzione hash/2

- Esempio: sia |T|=5 e h(k)=k mod 5



In pratica: si cerca di avere indipendenza dai dati

Paradosso del compleanno

- scelti 23 individui a caso, la probabilità che due di essi abbiano lo stesso compleanno è > ½ (50.7%)
  - dimostrare!
- in termini di hashing, anche ipotizzando di avere le pseudochiavi a distribuzione uniforme, dopo 22 inserimenti in una tabella di 365 celle, ogni ulteriore inserimento avrà probabilità superiore a ½ di generare una collisione

Interpretazione delle chiavi

- Le chiavi non sono necessariamente numeri naturali
  - Es.: stringhe
  - Soluzione: associare a ciascuna chiave un intero
  - Modalità dipendono da insieme delle chiavi e applicazione

Esempio: stringhe

- Possibile metodo: associare a ciascun carattere il valore ASCII e alla stringa il numero intero ottenuto in una base scelta
- Esempio: base 31, posizioni meno significative a destra

Stringa = "pt" → pseudochiave = $112*31^1+116*31^0=3588$

Ascii('p')=112

Ascii('t')=116

Derivazione di funzioni hash

- Molti metodi
  - Divisione
  - Ripiegamento
  - Mid-square
  - Estrazione
  - ....
- Obiettivo: distribuzione possibilmente uniforme
- Differenze:
  - Complessità
  - Fenomeni di agglomerazione

Divisione

- $h(k)=k \bmod |T|$ - Bassa complessità
- Attenzione ai fenomeni di agglomerazione
  - No potenze di 2: se $m=2^p$ allora tutte le chiavi con i p bit meno significativi uguali collidono
  - No potenze di 10 se le chiavi sono numeri decimali (motivo simile)
  - In generale, la funzione dovrebbe dipendere da tutte le cifre della chiave (comunque rappresentata)
  - Scelta buona in pratica: numero primo non troppo vicino a una potenza di 2 (esempio: $h(k)=k \bmod 701$ per $|K|=2048$ valori possibili)

Ripiegamento

- Chiave k suddivisa in parti $k_1,k_2,\ldots,k_n$
- $h(k)=f(k_1,k_2,\ldots,k_n)$
- Esempio: la chiave è un No. di carta di credito. Possibile funzione hash:

1. 4772 6453 7348 → {477, 264, 537, 348}
2. f(477,264,537,348) = (477+264+537+348)mod 701 = 224

Estrazione

- Si usa soltanto una parte della chiave per calcolare l'indirizzo
- Esempio: 6 cifre centrali del numero di carta di credito

4772 6453 7348 → 264537

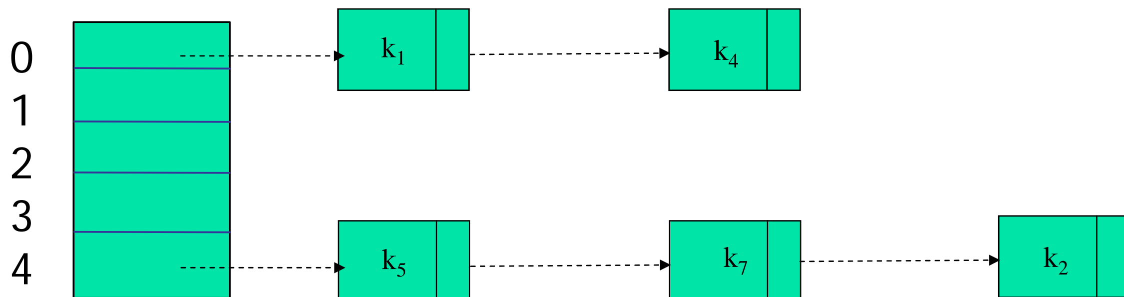
- Il numero ottenuto può essere ulteriormente manipolato
- L'indirizzo può dipendere da una porzione della chiave

Risoluzione delle collisioni

- I metodi si distinguono per la collocazione degli elementi che danno luogo alla collisione
- Concatenazione: alla i-esima posizione della tabella è associata la lista degli elementi tali che h(k)=i
- Indirizzamento aperto: tutti gli elementi sono contenuti nella tabella

Concatenazione

- $h(k_1)= h(k_4)=0$
- $h(k_5)= h(k_7)=h(k_2)=4$



- Es.: h(k)=k mod 5
- $k_1=0$, $k_4=10$
- $k_5=9$, $k_7=14$, $k_2=4$

Concatenazione/2

- insert(el, k): inserimento in testa alla lista associata alla posizione h(k) – costo $O(1)$
- search(k): ricerca lineare nella lista associata alla posizione h(k) – costo O(lungh. lista associata a h(k))
- delete(k): ricerca nella lista associata a h(k), quindi cancellazione – costo O(lungh. lista associata a h(k))

Indirizzamento aperto

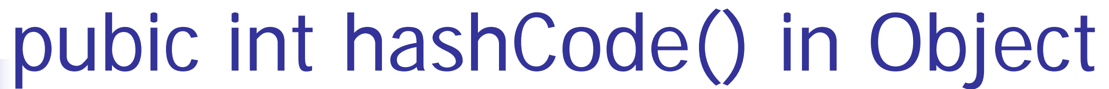
- Tutti gli elementi sono memorizzati nella tabella
- Le collisioni vanno risolte all'interno della tabella
  - Se la posizione calcolata è già occupata occorre cercarne una libera
  - I diversi metodi ad indirizzamento diretto si distinguono per il metodo di **scansione** adottato
  - La funzione hash può dipendere anche dal numero di tentativi effettuati
  - Indirizzo=h(k, i) per l'i-esimo tentativo

pubic int hashCode() in Object

- Definita in Object e quindi presente in tutte le classi Java
- Definita in Object in modo da generare un intero a partire dall'indirizzo dell'oggetto di invocazione
- Si noti che

  se obj1.equals(obj2) allora
  obj1.hashCode()==obj2.hashCode()
- Se equals() è stato ridefinito, allora hashCode() va necessariamente ridefinito affinché valga la condizione sopra
- Questo è fatto per tutte le libreria standard Java…
- … e va fatto in tutte le classi che definiremo noi!

pubic int hashCode() in Object



- … si noti che spesso nelle librerie Java condizioni

  (obj1.equals(obj2))

- Sono sostituite dalla condizione equivalente (se rispettiamo la regola sopra)

  (obj1.hashCode()==obj2.hashCode() && obj1.equals(obj2))

- Questo avviene per esempio nelle classi che realizzano Set e Map nel CollectionFramework