

# **Dizionari**

## **Specifiche, rappresentazione e confronto tra realizzazioni alternative.**

**Algoritmi e Strutture Dati + Lab**

A.A. 14/15

Informatica

Università degli Studi di Bari “Aldo Moro”

Nicola Di Mauro

# Introduzione

- Esistono delle applicazioni che, pur richiedendo una struttura dati del tipo insieme, non richiedono tutte le operazioni definite sull'insieme
  - Ad esempio, nel trattamento dei testi sono presenti spesso dei dizionari della lingua messi a disposizione per controllare la correttezza ortografica dello scritto. E' certamente un problema adatto al tipo astratto di dati insieme, ma non sembrano necessarie operazioni come unione, intersezione e differenza.
- Questi sottotipi del tipo insieme sono i **dizionari**
- Gli elementi sono generalmente tipi strutturati ai quali si accede per mezzo di un riferimento a un campo chiave. Gli elementi assumono la forma di una coppia costituita da (chiave, valore).

# Introduzione /2

- La caratteristica della chiave è legata alla applicazione:
  - nei dizionari degli elaboratori di testi la chiave individua una parola mentre in un dizionario di magazzino la chiave può essere un codice pezzo
- Il valore associato, invece, rappresenta l'informazione associata per scopi di gestione o manutenzione
- Esempio:
  - nelle tabelle di simboli, utili ai compilatori e ai linker, la chiave viene usata per l'identificatore mentre nel valore vengono memorizzate le informazioni di gestione (indirizzo della locazione, dimensione, posizioni nelle quali l'identificatore è usato nel programma etc.)

# Operazioni

- Le operazioni applicate ad un dizionario devono consentire la verifica dell'esistenza di una definita chiave e deve essere possibile l'inserimento di nuove coppie (chiave, valore) come pure la cancellazione.
- Può essere utile anche il recupero delle informazioni presenti nell'attributo oppure la loro eventuale modifica.
- Poiché possiamo definirli come un caso particolare di insieme, la specifica per i dizionari è identica a quella del tipo di dato insieme. Le operazioni ammesse sono:
  - crea, appartiene, inserisci, cancella
- In alcuni casi troviamo anche operazioni come
  - recupera e aggiorna

# Specifica sintattica

- Tipi
  - dizionario, boolean, chiave, valore
- Operatori
  - creadizionario:  $() \rightarrow \text{dizionario}$
  - dizionariovuoto:  $(\text{dizionario}) \rightarrow \text{boolean}$
  - appartiene:  $(\text{chiave}, \text{dizionario}) \rightarrow \text{boolean}$
  - inserisci:  $(\langle \text{chiave}, \text{valore} \rangle, \text{dizionario}) \rightarrow \text{dizionario}$
  - cancella:  $(\text{chiave}, \text{dizionario}) \rightarrow \text{dizionario}$
  - recupera:  $(\text{chiave}, \text{dizionario}) \rightarrow \text{valore}$

# Specifica semantica

- Tipi

- Dizionario = famiglia di dizionari costituita da coppie di tipo  $\langle \text{chiave}, \text{valore} \rangle$
- boolean = insieme valori verità

- Operatori

- $\text{creadizionario} = D$ 
  - $\text{post} : D = \{\}$
- $\text{dizionariovuoto}(D) = b$ 
  - $\text{post} : b = \text{vero se } D = \{\}, b = \text{falso altrimenti}$
- $\text{appartiene}(k, D) = b$ 
  - $\text{post} : b = \text{vero se esiste una coppia } \langle k', v \rangle \in D \text{ tale che } k' = k, b = \text{falso altrimenti}$
- $\text{inserisci}(\langle k, v \rangle, D) = D'$ 
  - $\text{post} : D' = D \cup \{ \langle k, v \rangle \}$  se non esiste una coppia  $\langle k', v' \rangle \in D$  tale che  $k' = k$ ;  
 $D' = D \setminus \{ \langle k', v' \rangle \} \cup \{ \langle k, v \rangle \}$  se esiste già una coppia  $\langle k', v' \rangle \in D$  tale che  $k' = k$ ;
- $\text{cancella}(k, D) = D'$ 
  - pre: esiste una coppia  $\langle k', v' \rangle \in D$  tale che  $k' = k$
  - $\text{post} : D' = D \setminus \{ \langle k', v' \rangle \}$
- $\text{recupera}(k, D) = v$ 
  - pre: esiste una coppia  $\langle k', v' \rangle \in D$  tale che  $k' = k$
  - $\text{post} : v = v'$

# Rappresentazioni

- Oltre alle realizzazioni viste per l'insieme, che si rifanno alla rappresentazione con vettore booleano (vettore caratteristico) e alla rappresentazione mediante una lista (i cui elementi sono quelli dell'insieme), ci sono realizzazioni più efficienti mediante vettori ordinati e tabelle hash.
- Rappresentazione con vettore ordinato
  - si utilizza un vettore con un cursore all'ultima posizione occupata
  - avendo definito una relazione di ordinamento totale  $\leq$  sulle chiavi, queste si memorizzano nel vettore in posizioni contigue e in ordine crescente a partire dalla prima posizione. Per verificare l'appartenenza di un elemento o chiave  $k$ , si utilizza la ricerca binaria (dicotomica, logaritmica), si confronta cioè il valore da ricercare  $k$  con il valore  $v$  che occupa la posizione centrale del vettore e si stabilisce in quale metà continuare la ricerca.

# Ricerca binaria

- Definizione dei tipi:
  - Dizionario: tipo strutturato con componenti
    - elementi: array di maxlung elementi di tipo tipoelem
    - ultimo: intero in  $[0..maxlung]$

```
bool APPARTIENE(k:tipoelem; D:dizionario per riferimento)
    return RICBIN(D.elementi, k, 1, D.ultimo)
```

```
bool RICBIN(V: vettore per riferimento; k: tipoelem; i: integer; j: integer)
    if i>j then
        RICBIN = false
    else
        m = (i + j) div 2
        if k = V[m] then
            RICBIN = true
        else
            if k < V[m] then
                RICBIN = RICBIN (V, k, i, m-1)
            else
                RICBIN = RICBIN (V, k, m+1, j)
```



# Hash

- Esiste una tecnica denominata “hash”, che si appoggia su di una struttura di dati tabellare, che si presta ad essere usata per realizzare dizionari.
  - Con questa struttura, le operazioni di ricerca e di modifica di un dizionario possono operare in tempi costanti e indipendenti sia dalla dimensione del dizionario che dall'insieme dei valori che verranno gestiti.
- Rappresentazione con tabella hash
  - idea base: ricavare la posizione che la chiave occupa in un vettore dal valore della chiave.
  - esistono diverse varianti che comunque si possono far risalire ad una forma statica e ad una forma dinamica o estensibile.
  - la prima fa uso di strutture o tabelle di dimensione prefissata, mentre l'hash dinamico è in grado di modificare dinamicamente le dimensioni della tabella hash sulla base del numero di elementi che vengono via via inseriti o eliminati. Nel seguito si farà riferimento solo alla forma statica.

# Rappresentazione con tabella Hash

- L'hash (statico) può assumere a sua volta due forme diverse denominate rispettivamente
  - **hash chiuso**: consente di inserire un insieme limitato di valori in uno spazio di dimensione fissa
  - **hash aperto**: consente di memorizzare un insieme di valori di dimensione qualsiasi in uno spazio potenzialmente illimitato
- Ambedue queste varianti però utilizzano una sottostante tabella hash a dimensione fissa costituita da una struttura allocata sequenzialmente in memoria e che assume la forma di un array

# Rappresentazione con tabella Hash /2

- Nel caso di hash chiuso la struttura sarà composta da un certo numero (maxbucket) di contenitori di uguale dimensione denominati bucket.
- Ognuno di questi contenitori può mantenere al proprio interno al massimo un numero  $nb = 1$  di elementi che comprenderanno la chiave e il corrispondente valore ( nel caso  $nb=1$  ogni bucket avrà una sola coppia (chiave, valore)
- Nel caso di hash aperto la struttura sarà composta da un certo numero indeterminato di contenitori bucket
- In ambedue i casi viene usata una funzione aritmetica allo scopo di calcolare, partendo dalla chiave, la posizione in tabella delle informazioni contenute nell'attributo collegato alla chiave.

# Rappresentazione con tabella Hash /3

- È una alternativa efficace all'indirizzamento diretto in un vettore perché la dimensione è proporzionale al numero di chiavi attese.
- Se  $K$  è l'insieme di tutte le possibili chiavi distinte e  $v$  è il vettore di dimensione  $m$  in cui si memorizza il dizionario, la soluzione ideale è la funzione di accesso  $h: K \rightarrow \{1, \dots, m\}$  che permetta di ricavare la posizione  $h(k)$  della chiave  $k$  nel vettore  $v$  così che, se  $k_1 \in K$  e  $k_2 \in K$ ,  $k_1 \neq k_2$  si ha  $h(k_1) \neq h(k_2)$ .
- Utilizzando  $m = |K|$  si ha garanzia di biunivocità e di poter accedere direttamente alla posizione contenente la chiave.
- Se  $|K|$  è grande, si ha spreco enorme di memoria.
  - la dimensione  $m$  del vettore va scelta in base al numero di chiavi attese.
- La soluzione di compromesso è scegliere un  $m$  maggiore di 1 ma molto minore di  $|K|$

# Rappresentazione con tabella Hash /4

- **Hash chiuso**

albinoni  $h(\text{albinoni}) = 1$

offenbach  $h(\text{offenbach}) = 15$

palestrina  $h(\text{palestrina}) = 16$

puccini  $h(\text{puccini}) = 16$

prokofev  $h(\text{prokofev}) = 16$

rossini  $h(\text{rossini}) = 18$

- Ci sono collisioni per i cognomi che iniziano con la stessa lettera

- Esempio: sia  $k = \{\text{cognomi di musicisti}\}$  e si assuma  $m = 26$ . Una possibile funzione è  $h(k) = h$ ,  $1 \leq h \leq 26$ , se il carattere iniziale di  $k$  è la  $h$ -esima lettera dell'alfabeto inglese.  $h$  non è biunivoca.

ALBINONI	1
:	
OFFENBACH	15
PALESTRINA	16
PUCCINI	17
PROKOFEV	18
ROSSINI	19
:	
	26

# Collisioni

- Una collisione si verifica quando chiavi diverse producono lo stesso risultato della funzione. Esistono funzioni hash più o meno buone anche se le collisioni non si potranno mai evitare del tutto.
- Nell'esempio visto si è adottata una semplice strategia per la risoluzione delle collisioni (**scansione lineare**):
  - se  $h(k)$  per qualunque chiave  $k$  indica una posizione già occupata, si ispeziona la posizione successiva nel vettore. Se la posizione è piena si prova con la seguente e così via fino a trovare una posizione libera o trovare che la tabella è completamente piena.
- Una posizione libera può venire facilmente segnalata in fase di realizzazione da una chiave fittizia *libero*.
- Per la cancellazione è più semplice sostituire l'oggetto cancellato con una chiave fittizia *cancellato* che dovrebbe essere facilmente distinguibile dalle altre chiavi reali e dall'altra chiave fittizia *libero*.
- La strategia lineare può produrre nel tempo il casuale addensamento di informazioni in certi tratti della tabella (**agglomerati**), piuttosto che una loro dispersione.

# Collisioni /2

- Quale che sia la funzione hash adottata, deve essere prevista una strategia per gestire il problema degli agglomerati e delle collisioni. In definitiva:
  - occorre una funzione hash, calcolabile velocemente e che distribuisca le chiavi uniformemente in  $v$ , in modo da ridurre le collisioni;
  - occorre un metodo di scansione per la soluzione delle collisioni utile a reperire chiavi che hanno trovato la posizione occupata e che non provochi la formazione di agglomerati di chiavi;
  - la dimensione  $m$  del vettore  $v$  deve essere una sovrastima del numero delle chiavi attese, per evitare di riempire  $v$  completamente.
- Per definire funzioni hash, è conveniente considerare la rappresentazione binaria  $\text{bin}(k)$  della chiave  $k$ .
  - Nel caso di chiavi non numeriche si può considerare la rappresentazione binaria dei caratteri e  $\text{bin}(k)$  è data dalla loro concatenazione.

# Collisioni /3

- 4 buoni metodi di generazione hash
  - denotiamo con  $\text{int}(b)$  il numero intero rappresentato da una stringa binaria  $b$ . Indichiamo da 0 a  $m-1$  gli elementi di  $v$ .
  - 1)  $h(k)=\text{int}(b)$ , dove  $b$  è un sottoinsieme di  $p$  bit di  $\text{bin}(k)$ , solitamente estratti nelle posizioni centrali
  - 2)  $h(k)=\text{int}(b)$ , dove  $b$  è dato dalla somma modulo 2, effettuata bit a bit, di diversi sottoinsiemi di  $p$  bit di  $\text{bin}(k)$ .
  - 3)  $h(k)=\text{int}(b)$ , dove  $b$  è un sottoinsieme di  $p$  bit estratti dalle posizioni centrali di  $\text{bin}(\text{int}(\text{bin}(k))^2)$ .
  - $h(k)$  è uguale al resto della divisione  $\text{int}(\text{bin}(k)) / m$  ( $m$  è dispari; se fosse uguale a  $2^p$ , due numeri con gli stessi  $p$  bit finali darebbero sempre luogo a una collisione).
- L'ultima funzione hash definita è la migliore dal punto di vista probabilistico e fornisce un'eccellente distribuzione degli indirizzi  $h(k)$  nell'intervallo  $[0, m-1]$ .



# Collisioni /4

## Esempio

- **Funzione hash:** si supponga che le chiavi siano di 6 caratteri alfanumerici (chiavi più lunghe sono troncate, mentre chiavi più corte sono espanse a destra con spazi). Si assuma  $\text{ord}(a)=1$ ,  $\text{ord}(b)=2$ , ...,  $\text{ord}(z)=26$  e  $\text{ord}(b)=32$ , dove  $b$  indica lo spazio, con rappresentazione di ogni ordinale su 6 bit.
- Per le chiavi weber e webern si ottiene:
  - $\text{bin}(\text{weber}) = 010111\ 000101\ 000010\ 000101\ 010010\ 100000\ 0000$
  - $\text{bin}(\text{webern}) = 010111\ 000101\ 000010\ 000101\ 010010\ 001110\ 0000$
- dove si sono evidenziati per chiarezza i gruppi di 6 bit che rappresentano ciascun carattere. Calcoliamo gli indirizzi hash ottenuti con le funzioni definite di seguito in (a), (b) e (d):
  - (a) sia  $m = 256 = 2^8$ . Estruendo gli 8 bit dalla posizione 15 alla 22 di  $\text{bin}(k)$ ,  $h(\text{weber}) = h(\text{webern}) = \text{int}(00100001) = 33$ . Le due chiavi danno pertanto luogo ad una collisione

# Collisioni /5

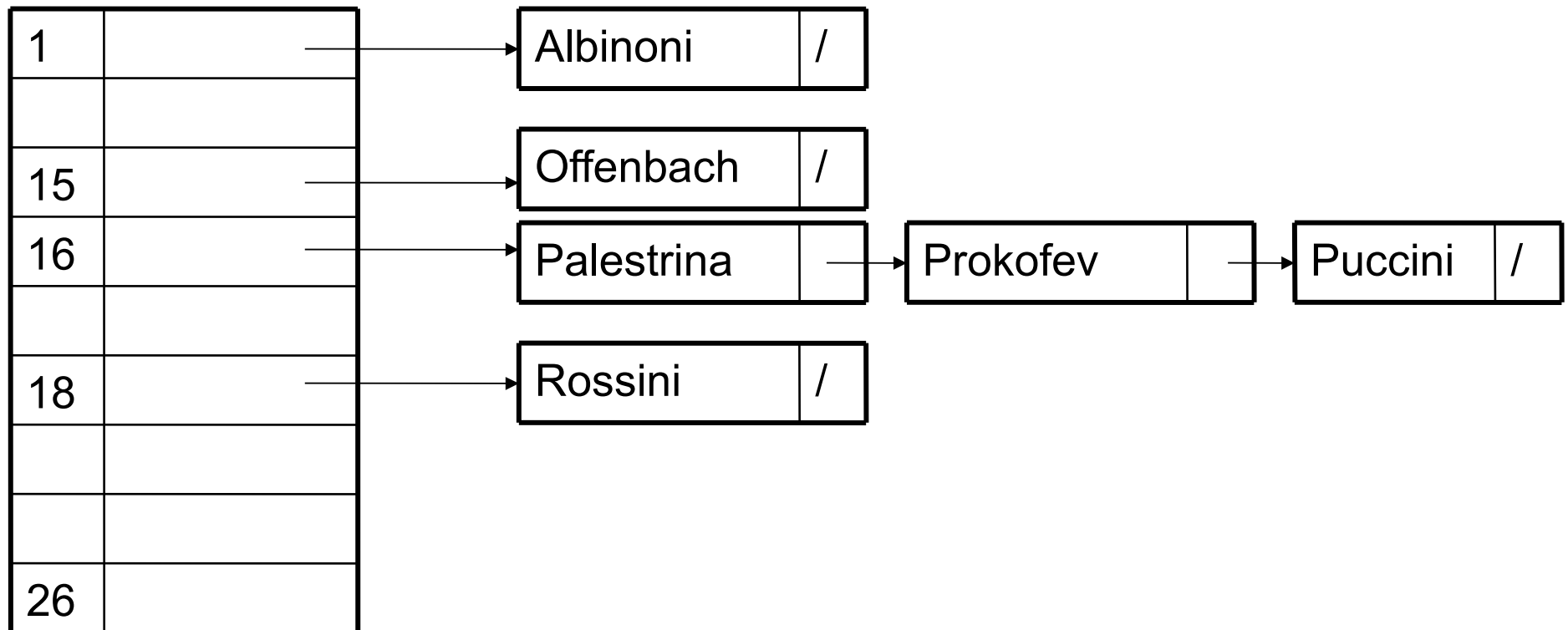
- (b) sia ancora  $m = 256 = 2^8$  e si calcoli  $b$  raggruppando  $\text{bin}(k)$  in cinque gruppi di 8 bit (dopo aver espanso a destra  $\text{bin}(k)$  con quattro zeri).  $h(\text{weber}) = \text{int}(11000011) = 195$ , poiché
  - $11000011 = 01011100 + 01010000 + 10000101 + 01001010 + 00000000$
- dove  $+$  indica la somma bit a bit modulo 2. Analogamente,  $h(\text{webern}) = \text{int}(00100001) = 33$ , poiché
  - $00100001 = 01011100 + 01010000 + 10000101 + 01001000 + 11100000$
- benché  $h(\text{webern})$  sia uguale a 33, come nel caso (a), la collisione è stata eliminata.
- (d) sia  $m = 383$ . utilizzando 6 bit,  $\text{int}(\text{bin}(k))$  è un numero in base  $2^6 = 64$ .
- $\text{int}(\text{bin}(\text{webern})) = 23 * 64^5 + 5 * 64^4 + 2 * 64^3 + 5 * 64^2 + 18 * 64^1 + 14 * 64^0 = 64(64(64(64(23*64)+5)+2)+5)+18)+14$
- $h(\text{webern})$  e  $h(\text{weber})$  sono ottenuti prendendo il resto della divisione  $\text{int}(\text{bin}(\dots))/383$

# Hash aperto

- Una tecnica che evita la formazione di agglomerati è quella dell'hash aperto che richiede che la tabella hash mantenga la lista degli elementi le cui chiavi producono lo stesso valore di funzione (trasformata).
- La tabella hash viene realizzata definendo un array di liste di bucket (liste di trabocco).
- La funzione hash viene utilizzata per determinare quale lista potrebbe contenere l'elemento che possiede una determinata chiave in modo da poter attivare una successiva operazione di ricerca nella lista corrispondente e da restituire la posizione del bucket che contiene la chiave.

# Liste di trabocco

- Il vettore  $v$  contiene in ogni posizione un puntatore ad una lista



# Metodi di scansione

- Nella realizzazione con hash e liste di trabocco si è usato un metodo di scansione esterno contrapposto alla scansione lineare che è un metodo di scansione interno.
- Altri metodi interni (o chiusi) sono
  - scansione quadratica
  - scansione pseudocasuale
  - hashing doppio
- **scansione interna**
  - chiamiamo  $f_i$  la funzione che viene utilizzata l' $i$ -esima volta che si trova occupata una posizione del vettore  $v$ ,  $i \geq 0$ , (per  $i=0$ ,  $f_0 = h$ ).
  - $f_i$  va scelta in modo da toccare tutte le posizioni di  $v$  (una sola volta).

# Metodi di scansione /2

- **Scansione lineare**

- $f_i = (h(k) + h*i) \bmod m$
- $h$  è un intero positivo primo con  $m$
- $h$  rappresenta la distanza tra due posizioni successive esaminate nella scansione (se  $h = 1$ , scansione a passo unitario).
- essendo  $h$  e  $m$  primi tra loro, vengono esaminate tutte le posizioni di  $v$  prima di riconsiderare le posizioni già esaminate.
- **svantaggio:** non riduce la formazione di agglomerati

# Metodi di scansione /3

- **Scansione quadratica**

- $f_i = (h(k) + h*i + i(i-1)/2) \bmod m$
- $m$  e' primo
- la distanza tra due posizioni successive nella sequenza è variabile, quindi la possibilità di agglomerati è ridotta. Svantaggio: la sequenza di scansione non include tutte le posizioni di  $v$  (svantaggio trascurabile per  $m$  non troppo piccolo).

- **Scansione pseudocasuale**

- $f_i = (h(k) + p_i) \bmod m$
- $p_i$  è l' $i$ -esimo numero generato da un generatore di numeri pseudocasuali, che genera gli interi tra 1 e  $m$  una sola volta in un ordine qualunque.

- **Hashing doppio**

- $f_i = (h(k) + i * f(k)) \bmod m$
- $f$  è un'altra funzione hash diversa da  $h$ .

# Metodi di scansione /4

- Usando metodi di scansione interna e potendo cancellare chiavi, non si è mai sicuri che, raggiunta una posizione vuota nella ricerca di  $k$ , tale chiave non si trovi in un'altra posizione di  $v$ , poiché la posizione ora vuota era occupata quando  $k$  è stata inserita.
- Bisogna dunque scandire anche le posizioni in cui si è cancellato e fermarsi o sulla posizione mai riempita o dopo essere tornati su una posizione già scandita.
- Ciò determina un aumento del tempo di ricerca.
- Utilizzare il metodo di scansione esterna se sono previste molte cancellazioni.