Dizionario

I **dizionari** sono dei sottotipi del tipo insieme dove 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>`.

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.
Le operazioni ammesse su un dizionario sono **crea**, **appartiene**, **inserisci**, **cancella**, **recupera** e **aggiorna**.

Specifica sintattica

Tipi: `dizionario, boolean, chiave, valore`
Operatori:
- `creadizionario: () → dizionario`
- `dizionariovuoto: (dizionario) → boolean`
- `appartiene: (chiave, dizionario) → boolean`
- `inserisci: (<chiave, valore>, dizionario) → dizionario`
- `cancella: (chiave, dizionario) → dizionario`
- `recupera: (chiave, dizionario) → valore`

Specifica semantica

Tipi:
- `dizionario`: famiglia di dizionari costituita da coppie di tipo <chiave, valore>

- `boolean`: insieme valori verità

Operatori:

- `creadizionario = D`
  - Post: D = { }
- `dizionariovuoto(D) = b`
  - Post: b = true se D = { }
    b = false altrimenti
- `appartiene(k, D) = b`
  - Post: b = true se ∃ <k', v> ∈ D tale che k' = k
    b = false altrimenti
- `inserisci(<k, v>, D) = D'`
  - Post: D' = D U {<k, v>} se ∄ <k', v'> ∈ D tale che k' = k
    D' = D \ {<k, v>} U {<k, v>} se ∃ <k', v'> tale che k' = k
- `cancella(k, D) = D'`
  - Pre: ∃ <k', v'> ∈ D tale che k' = k
  - Post: D' = D \ {<k', v'>}
- `recupera(k, D) = v`
  - Pre: ∃ <k', v'> ∈ D tale che k' = k
  - Post: v = v'

Rappresentazioni

Oltre alle realizzazioni viste per l'insieme, che si rifanno alla rappresentazione con **vettore booleano** (detto anche **vettore caratteristico**) e alla rappresentazione mediante una **lista**, 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** <= 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** (anche detta **logaritmica**): si confronta 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.

```
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, j: integer)
if i > j then
      return false
else
      m = (i + j)/2
      if k = V[m] then
            return true
      else
```

```
if k < V[m] then
      ricbin = ricbin(V, k, i, m-1)
else
      ricbin = ricbon(V, k, m+1, j)
```

Hash

Esiste una tecnica denominata “**hash**”, che si appoggia su di una struttura 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

L’idea di base è quella di **ricavare la posizione** che occupa la chiave in un vettore, **dalla chiava stessa**. Esistono diverse varianti che comunque si possono far risalire ad una forma **statica** e ad una forma **dinamica**:

- La prima fa uso di tabelle di dimensione **prefissata**;
- La seconda è in grado di modificare **dinamicamente** le dimensioni della tabella hash sulla base del numero di elementi che vengono via via inseriti o eliminati.

L’**hash statico** può assumere a sua volta due forme diverse denominate rispettivamente:

- Hash **chiuso**, che consente di inserire un insieme limitato di valori in uno spazio di dimensione fissa;
- Hash **aperto**, che consente di memorizzare un insieme di valori di dimensioni qualsiasi in uno spazio potenzialmente illimitato.

Entrambe le varianti utilizzano una **tabella hash** a dimensione fissa costituita da una struttura allocata sequenzialmente in memoria e che assume la forma di un array.
Nel caso di **hash chiuso**, la struttura sarà composta da un certo numero, indicato da `maxbucket`, di contenitori di uguale dimensione denominati appunto **bucket**. Ognuno di questi contenitori può mantenere al proprio interno al massimo un numero `nb` di elementi che comprenderanno la chiave e il corrispondente valore. Ad esempio, se `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 entrambi 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. Questa è una alternativa efficace all’indirizzamento diretto in un vettore perché la dimensione è proporzionale al numero di chiavi attese. Indichiamo con:

- `k`, l’insieme di tutte le possibili **chiavi** distinte;
- `v`, il **vettore** di dimensione `m` in cui si memorizza il dizionario

La soluzione ideale è la funzione di accesso h: K → {1, …, m} che permette di ricavare la posizione `h(k)` della chiave `k` nel vettore `v` così che:

- Se $k_1 \in K$ e $k_2 \in K$
- Con $k_1 \neq k_2$
    - Si ha $h(k_1) \neq h(k_2)$

Utilizzando m = |K| si ha la garanzia di binuvocità e di poter accedere direttamente alla posizione contenente la chiave. Se |K| è grande, si ha però **spreco** enorme **di memoria**. La dimensione m del vettore va scelta in base al numero di chiavi attese. La soluzione di compromesso è di scegliere 1 ≤ m ≤ |K|.

Hash chiuso

Sia `k = {cognomi di musicisti}` e si assuma `m = 26`.

albinoni h(albinoni) = 1
offenbach h(offenbach) = 15
palestrina h(palestrina) = 16
puccini h(puccini) = 16
prokofev h(prokofev) = 16
rossini h(rossini) = 18

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

Una possibile funzione è `h(k) = h` con 1 ≤ `h` ≤ 26, se il carattere iniziale di `k` è la `h`-esima lettera dell'alfabeto inglese `h` non è biunivoca. In questo modo, ci sono **collisioni** per i cognomi che iniziano con la stessa lettera, e la funzione hash utilizzata non va bene.

Collisioni

Una **collisione** si verifica quando chiavi diverse producono lo **stesso risultato** della funzione. Esistono funzioni hash più o meno buona 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, chiamata 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 "capire" 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, che prendono il nome di **agglomerati**, piuttosto che una loro dispersione. 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 trovate 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 di chiavi attese, per evitare di riempire `v` completamente.

Per definire le funzioni hash, è conveniente considerare la **rappresentazione binaria** `bin(k)` della chiave `k`.
Alcuni metodi di generazione hash b = bin(k) sono i seguenti:
Denotiamo con `int(b)` il numero intero rappresentato da una **stringa binaria** `b`.
Indichiamo da 0 a `m-1` gli elementi di `v`.

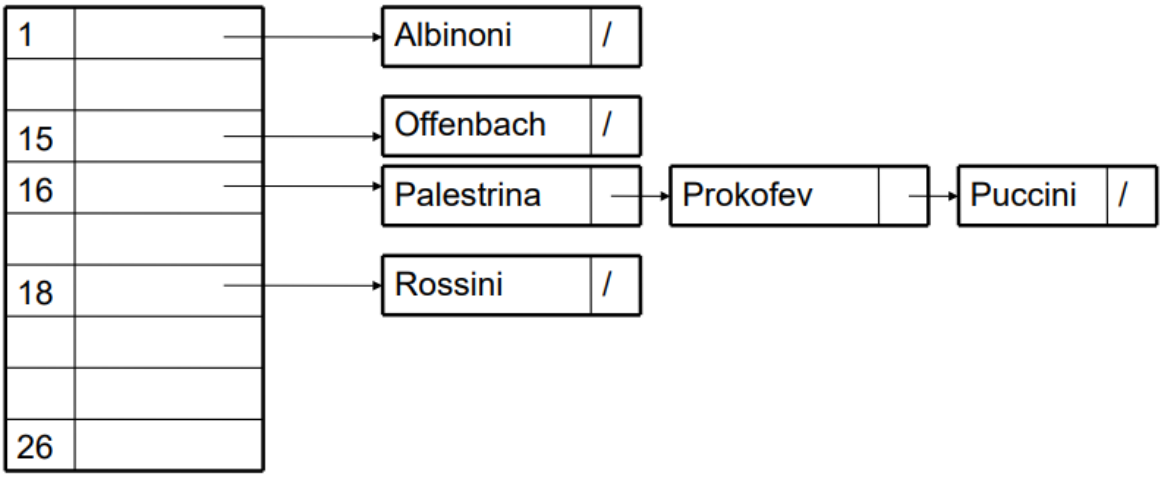
- `h(k) = int(b)`, dove `b` è un sottoinsieme di `p` bit di `bin(k)`, solitamente estratti nelle posizioni centrali;
- `h(k) = int(b)`, dove `b` è dato dalla somma modulo 2, effettuata bit a bit, di diversi sottoinsiemi di `p` bit di `bin(k)`;
- `h(k) = ⌊m * (i * C - ⌊i * C⌋)⌋`, dove `m` è un numero di potenza 2 e `C` è un numero reale compreso tra 0 e 1 mentre `i = int(bin(k))`;
- `h(k) = resto della divisione int(bin(k)) / m`, con `m` dispari; in questo caso se fosse uguale a `2p`, due numeri con gli stessi `p` bit finali darebbero sempre luogo ad 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]`.

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**, denominate **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. 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 sono la scansione quadratica, la scansione pseudocasuale e l'hashing doppio.

- **Scansione interna**: chiamiamo f_i la funzione che viene utilizzata per l'`i`-esima volta che si trova occupata una posizione del vettore `v`, con $i \geq 0$;
  - Per $i = 0$ si ha che $f_0 = h$; f_i va scelta in modo da toccare tutte le posizioni di `v` una sola volta;
- **Scansione lineare**: $f_i = (h(k) + h * i) \bmod m$
  - `h` è un intero positivo primo con `m` e rappresenta la distanza tra due posizioni successive esaminate nella scansione. Se $h = 1$, la scansione avviene a passo unitario. Essendo h e m primi tra loro, vengono esaminate tutte le posizioni di `v` prima di riconsiderare le posizioni già esaminate;
  - Ha lo **svantaggio** di non ridurre la formazione di agglomerati;
- **Scansione quadratica**: $f_i = (h(k) + h * i^2) \bmod m$
  - `m` è primo e la distanza tra due posizioni successive nella sequenza è variabile, quindi la possibilità di agglomerati è ridotta;
  - Ha lo **svantaggio** di non includere tutte le posizioni di v durante la sequenza di scansione, ma ciò è trascurabile per una m non troppo piccola;
- **Scansione pseudocasuale**: $f_i = (h(k) + r_i) \bmod m$
  - r_i è l'`i`-esimo numero generato da un generatore di numeri pseudocasuali, che genera gli interi tra 1 e `m` una solta volta in un ordine qualunque;
- **Hashing doppio**: $f_i = (h(k) + i * h'(k)) \bmod m$
  - `h'` è un'altra funzione di hash diversa da `h`.

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 sulle posizione mai riempita o dopo essere tornati su una posizione già scandita. Ciò determina

un **aumento del tempo di ricerca**. Se sono previste molte cancellazioni è meglio utilizzare il metodo di **scansione esterna** a quella interna.