

# 1 Hashing

L'**hashing** è una **tecnica alternativa** agli alberi binari di ricerca (capitolo 4.4) o agli array associativi, utilizzata per realizzare i dizionari.

A differenza dell'implementazione tramite alberi, nella quale si riusciva a mantenere la stessa complessità nei casi di `insert()`, `lookup()` e `remove()`, nel caso ideale, la cosa migliore sarebbe quella di **mantenere una complessità costante** per tutti i tipi di operazioni, che sia inoltre **inferiore** a quella delle strutture ad albero.

Questa implementazione ideale prende il nome di **tabelle di hash**.

	Array non ordinato	Array ordinato	Lista	Alberi RB	Impl. ideale Hash table
<code>insert()</code>	$O(1), O(n)$	$O(n)$	$O(1), O(n)$	$O(\log n)$	$O(1)$
<code>lookup()</code>	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$	$O(1)$
<code>remove()</code>	$O(n)$	$O(n)$	$O(n)$	$O(\log n)$	$O(1)$
<code>foreach</code>	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$\Theta(n)O(m)$

Per comprendere il funzionamento delle tabelle hash dobbiamo prendere in considerazione il concetto di **insieme universo**  $U$ , ovvero un insieme di tutte le possibili chiavi, la cui grandezza dell'insieme varia arbitrariamente in base ai dati che si vogliono contenere.

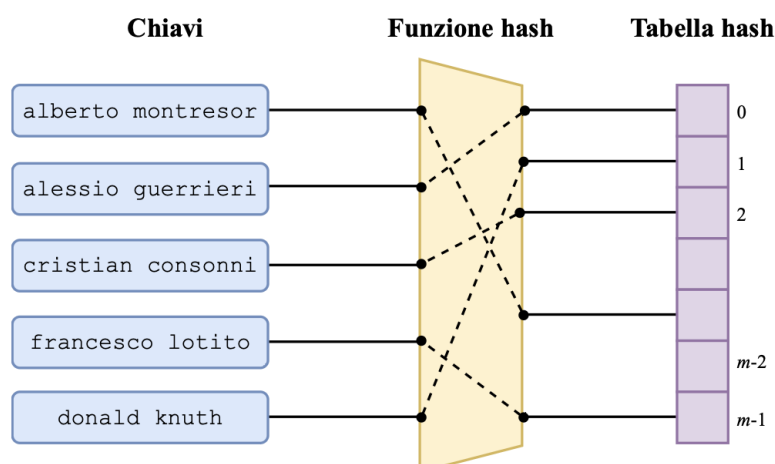
## Idea di base

Quello che si vuole fare è memorizzare tutti i dati dell'insieme  $U$  in un vettore di dimensione ( $m$ ) finita  $T[0 \dots m-1]$ , ed avere un meccanismo per cui, data una chiave, trovare rapidamente la posizione in cui è memorizzata.

Le chiavi possono essere delle stringhe, degli oggetti o dei numeri, e il compito delle tabelle hash è quello di trasformarle in un indice all'interno di esse. Per fare ciò vengono utilizzate le **funzioni hash**.

## Cos'è una funzione hash?

Una **funzione hash** è una funzione  $H$  che mappa ciascuna chiave  $k$  appartenente all'insieme universo  $U$  ( $k \in U$ ) nell'indice  $H(k)$  di un vettore  $A$ , destinato a contenere la coppia  $(k, v)$ . Viene definita come  $H : U \rightarrow \{0, 1, \dots, m-1\}$ .



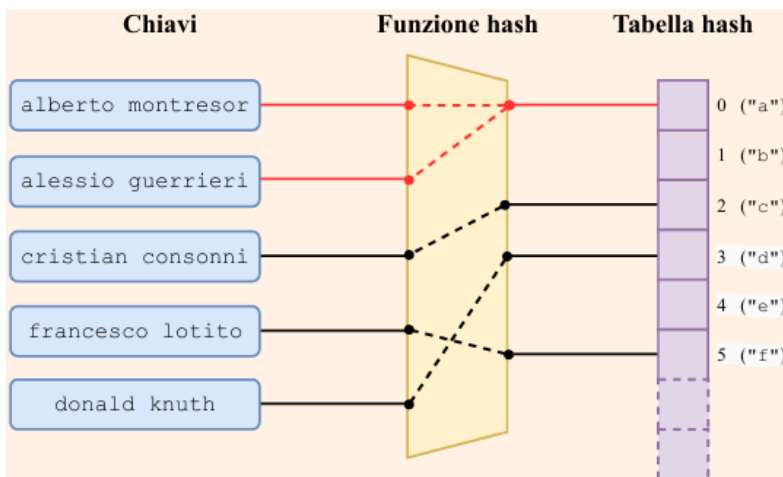
Come si può vedere dall'immagine, viene presa una chiave (in questo caso una stringa) che verrà poi trasformata in qualche modo, tramite la funzione hash (che può essere anche una black box), in un indice.

A questo punto sorge un **problema**: l'insieme delle chiavi è potenzialmente infinito, ma non si vuole che sia lo stesso anche per la tabella

hash, dal momento in cui si potrebbe non disporre dello spazio in memoria necessario per qualunque coppia chiave-valore. Dunque, quello che succede è che avvengono delle **collisioni**. Per evitare inutili sprechi di memoria, la dimensione  $m$  del vettore non deve essere determinata sulla base dell'intero universo  $U$ , ma piuttosto in funzione del **numero di chiavi attese**, ovvero la **quantità  $k$  di elementi** che si prevede saranno **effettivamente presenti nel dizionario** in un determinato momento.

#### Quando si verifica una collisione?

Si verifica una **collisione** quando due chiavi distinte  $k_1 \neq k_2$  vengono mappate dalla funzione hash sulla stessa posizione del vettore, ovvero quando  $H(k_1) = H(k_2)$ .



Le collisioni mettono quindi in luce due **aspetti fondamentali** nella progettazione di una tabella hash:

- La **scelta della funzione hash** è cruciale, infatti una funzione mal progettata può distribuire le chiavi in modo sbilanciato, causando una concentrazione eccessiva in alcune posizioni del vettore, lasciando quasi del tutto inutilizzate altre.
- È essenziale progettare dei **meccanismi** efficienti per la **gestione delle collisioni**, poiché anche scegliendo una buona funzione di hash, quando il numero di chiavi possibili supera il numero di posizioni disponibili nella tabella, le collisioni sono inevitabili. Questi meccanismi permettono quindi di **garantire il corretto funzionamento** delle operazioni di inserimento, ricerca e cancellazione.