Tabelle Hash

- Obiettivi:
  - Presentare le tabelle hash e la nozione di funzione di hashing
  - Discutere la complessità di questa realizzazione per le Tavole

Tabelle ad accesso diretto

- Sono implementazioni di dizionari (tavole) basati sulla proprietà di accesso diretto alle celle di un array
- Idea:
  - dizionario memorizzato in array V di m celle
  - a ciascun elemento è associata una chiave intera nell'intervallo [0,m-1]
  - elemento con chiave k contenuto in V[k]
  - al più n≤m elementi nel dizionario

Implementazione

classe `TavolaAccessoDiretto` **implementa** `Dizionario`:

dati: $S(m) = \Theta(m)$

un array v di dimensione $m \geq n$ in cui $v[k] = elem$ se c'è un elemento $elem$ con chiave k nel dizionario, e $v[k] =$ `null` altrimenti. Le chiavi k devono essere interi nell'intervallo $[0, m-1]$.

operazioni:

`insert`$(elem\ e, chiave\ k)$ $T(n) = O(1)$

$v[k] \leftarrow e$

`delete`$(chiave\ k)$ $T(n) = O(1)$

$v[k] \leftarrow$ `null`

`search`$(chiave\ k) \rightarrow elem$ $T(n) = O(1)$

return $v[k]$

Pro e contro

Pro:

- Tutte le operazioni richiedono tempo O(1)

Contro:

- Le chiavi devono essere necessariamente interi in [0, m-1]
- Lo spazio utilizzato è proporzionale al numero di chiavi (m) non al numero di elementi (n)
- Grande spreco di memoria!

Fattore di carico

- Il grado di riempimento di una tabella è infatti dato dal fattore di carico: num. elementi num. chiavi

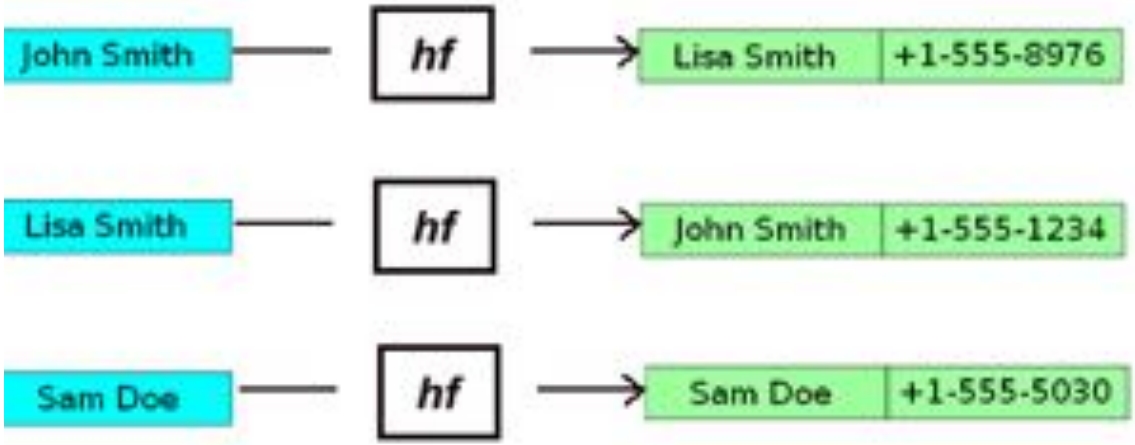
$$\alpha = n / m$$

- (n, numero di elementi della tavola; m, numero di chiavi dellarray)
- Esempio: tabella con nomi di cento (n=100) studenti indicizzati da numeri di matricola a 6 cifre, n=100 $m=10^6$ $\alpha = 0{,}0001 = 0{,}01\%$
- Si occupa solo lo 0,01% dell'array

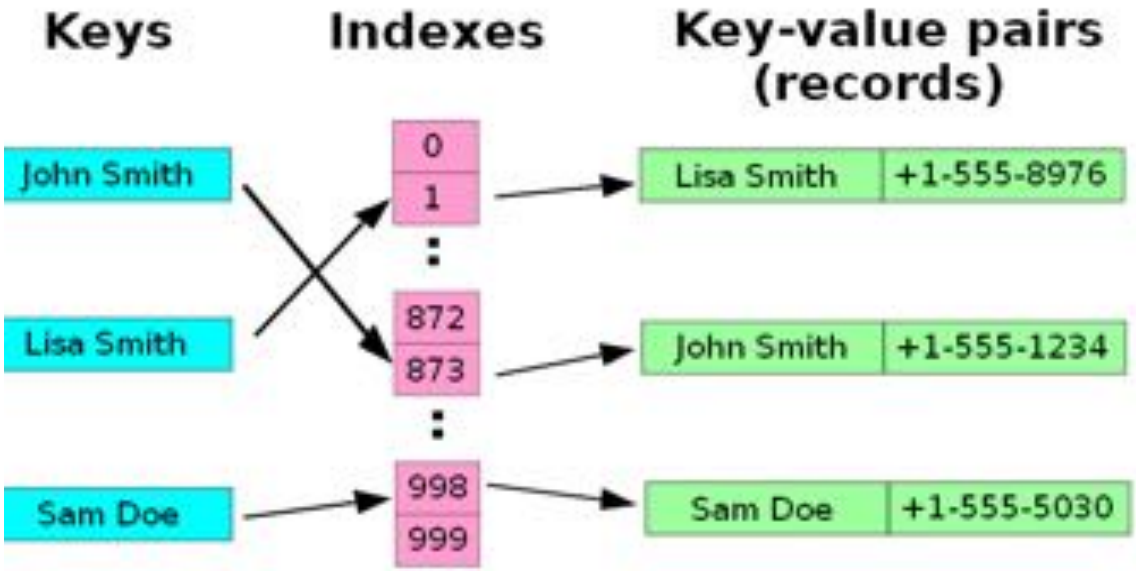
Tabelle hash (*hash map*)

- Fa corrispondere una data *chiave* con un dato *valore (indice)* attraverso una ***funzione hash***
- L'idea:
  - Chiavi prese da un universo totalmente ordinato U (possono anche non essere numeri)
  - Funzione hash: h: U $\rightarrow$ [0, m-1] (funzione che trasforma chiavi in indici)
  - Elemento con chiave k in posizione V[h(k)]
- Usate per l'implementazione di strutture dati associative astratte come **`Map`** o **`Set`** (vedi parte Java sulla JCF)

Mappe associative (tabelle hash)



Mappe basate su funzioni hash



Mappe basate su indici (strutture dati aggiuntive, spesso realizzate con alberi)

Funzioni Hash perfette

- Idealmente, chiavi diverse dovrebbero essere trasformate in indirizzi differenti:

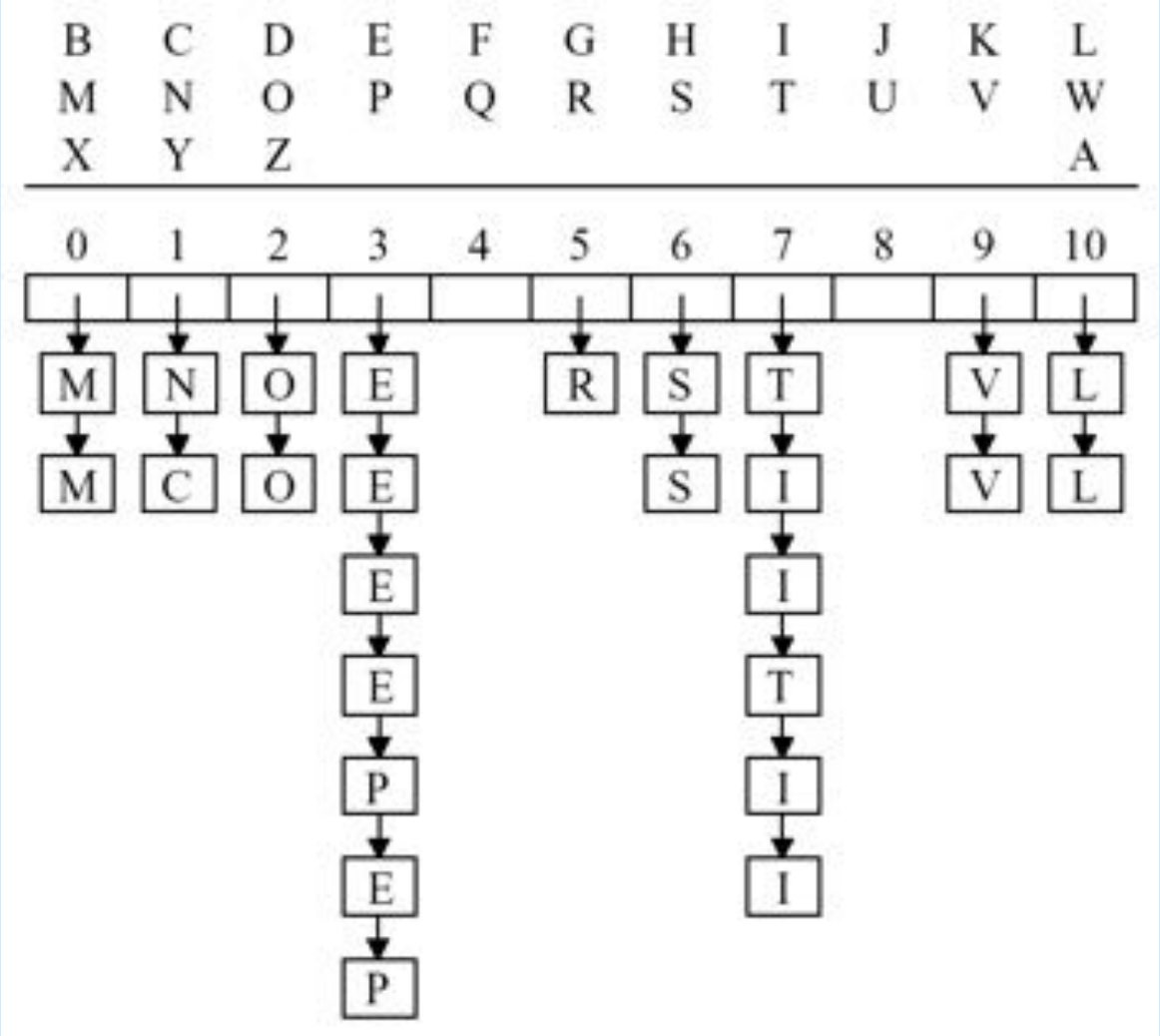
$$u \neq v \Rightarrow h(u) \neq h(v)$$

- La cardinalità dell'insieme delle possibili chiavi dovrebbe essere molto più piccolo del numero di chiavi da memorizzare: $|U| \leq m$
- Raramente praticabile
- Se la ***funzione hash non perfetta*** applicata a due chiavi diverse può generare il medesimo indirizzo, si ha ***collisione***

Esempio

- Tabella hash con elementi aventi come chiavi lettere dell'alfabeto U={A,B,C,…}
- Funzione hash non perfetta (ma buona in pratica per m primo): h(k) = ascii(k) mod m
- Ad esempio, per m=11: h('C') = h('N') $\Rightarrow$ se volessimo inserire sia 'C' and 'N' nel dizionario avremmo una collisione
- Gestione delle collisioni

Gestione delle collisioni



Esempio: attra-verso ***liste di collisione***

Ma possibili anche altri metodi (ad esempio, ***indirizzamento aperto***, passa alla casella successiva se libera)

Gestione delle collisioni

- ***Liste di collisione***, gli elementi in sono contenuti in liste esterne alla tabella: V[i] punta alla lista degli elementi tali che h(k)=i
- ***Indirizzamento aperto***, tutti gli elementi sono contenuti nella tabella: se una cella è occupata, se ne cerca un'altra libera
- ***Aumenta il costo delle operazioni*** (ricerca, cancellazione, inserimento) rispetto al caso ideale (O(1)) che si ha con funzioni hash perfette

Uniformità

- L'*hashing* è un problema classico dell'informatica (*hashing* utilizzato anche dai programmi di gestione di basi di dati)
- Per ridurre la probabilità di collisioni, una buona funzione *hash* dovrebbe essere in grado di ***distribuire in modo uniforme*** le chiavi nello spazio degli indici del vettore
- Non vedremo realizzazioni in C, ma ritroveremo le tavole *hash* nella Java Collection Framework (JCF)