Problema

Problema

Abbiamo un insieme S di elementi da memorizzare con $m = |S|$ posizioni

- Ogni elemento x ha una **chiave e dati satellite**
- Le **operazioni** necessarie sono inserimento, cancellazione e ricerca

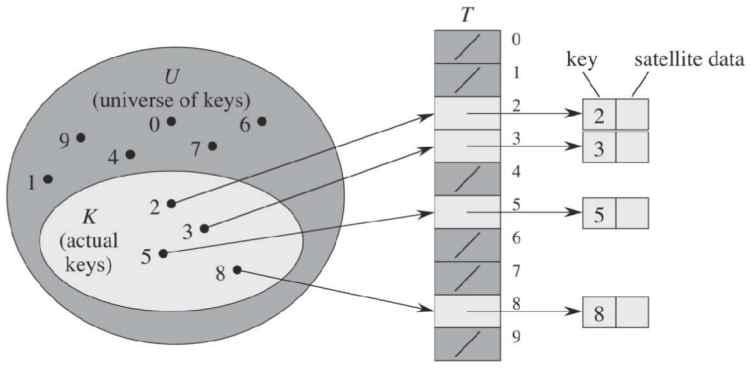
Tramite una corretta conversione, è sempre possibile trasformare una chiave di un qualsiasi tipo in un **valore numerico**.

Daremo quindi per scontato che la chiave sia trasformata quando necessario.

Indirizzamento Diretto

Idea

Utilizziamo un vettore T di grandezza m, possiamo salvare ogni dato nella posizione corrispondente alla chiave



Utilizziamo un vettore T (tabella) di puntatori con m posizioni (indicizzate da 0 a $m-1$), dove $T[x.key]$ punta all'elemento $< x.key, x.data >$

DIRECT-ADDRESS-SEARCH(T, k)

```
1   return T[k]
```

Parametri: *T=tabella, k=chiave da cercare*
Return: *puntatore all'elemento con chiave k*

Costo Ricerca

$\Theta(1)$

DIRECT-ADDRESS-INSERT(T, x)

```
1   T[x.key] = x
```

Parametri: *T=tabella, x=elemento da inserire*

Costo Inserimento

$\Theta(1)$

DIRECT-ADDRESS-DELETE(T, k)

```
T[k] = NIL
```

***Parametri**: T=tabella, k=chiave elemento da eliminare*

Costo Eliminazione

$$\Theta(1)$$

Dimensione Della Tabella

Quando lo spazio delle chiavi è molto più grande di quelle attive, la tabella presenterà molte celle vuote che sprecano memoria.

Inoltre, a volte l'insieme delle chiavi possibili è talmente grande che è impossibile creare tabelle di tali dimensioni

Hash Tables

Idea

Per risolvere il problema della dimensione della tabella, possiamo dare a T una dimensione limitata m e salvare ogni chiave in un indice tra 0 e $m-1$

Possiamo implementare questa idea tramite le **funzioni hash**

Funzioni Hash

Definizione: Funzione Hash

Definiamo funzione hash una funzione che prende in input una chiave e **restituisce un indice** compreso tra $[0, m-1]$

$$h : K \to [0, m-1]$$

Possiamo quindi implementare una **hash table** utilizzando un vettore T (tabella) con dimensione m dove per ogni elemento x viene salvato un puntatore all'indice dato dalla funzione h

$$T[h(x.key)] = x$$

Collisioni

Tramite le funzioni hash, due chiavi diverse potrebbero restituire lo stesso indice

$$k_i \neq k_j \wedge h(k_i) = h(k_j)$$

In questo caso, bisogna gestire le **collisioni**, tramite, per esempio, chaining o open addressing

A prescindere dal metodo di gestione, per ridurre le collisioni sarà necessaria una funzione hash il più **uniforme** possibile

Definizione: Funzione Hash Uniforme

Definiamo una funzione hash h uniforme se per ogni chiave k e per ogni coppia di indici i, j

$$\text{Prob}(h(k) = i) = \text{Prob}(h(k) = j)$$

La probabilità che una chiave dia un indice o un altro è la stessa

Esempio: Metodo Della Divisione

$$h(k) = k \bmod m$$

Esempio: Metodo Della Moltiplicazione

Sia $0 < A < 1$ una costante

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

Moltiplica m per un decimale tra $]0,1[$ e ne prende la parte intera inferiore

Funzione Hash Ideale

Il concetto di funzione hash uniforme è **puramente teorico**: per avvicinarsi il più possibile a questo concetto, una funzione hash deve essere

- **Deterministica**
- Assomigliare il più possibile ad una funzione **random**
- Considerare eventualmente la **struttura statistica** delle chiavi, ovvero la compensare la probabilità che venga passata una chiave con determinate caratteristiche piuttosto che altre

Load Factor

Definizione

Sia m la dimensione della tabella, n il numero di chiavi presenti

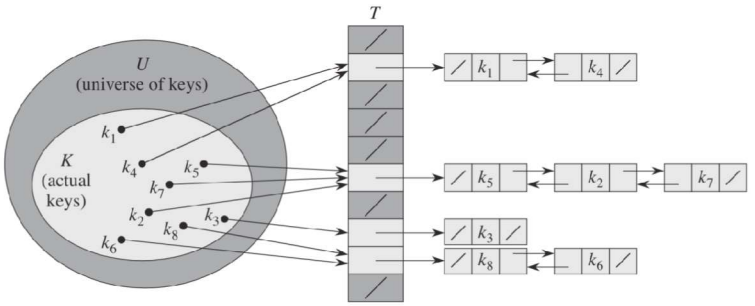
Definiamo load factor il rapporto

$$\alpha = \frac{n}{m}$$

Chaining

Idea

In ogni posizione della tabella, invece che salvare un singolo puntatore, salviamo una lista di nodi contenente tutti gli elementi che collidono su quell'indice



Operazioni

CHAINED-HASH-INSERT(T, x)

```
1   LIST-INSERT(T[h(x.key)], x)
```

Parametri: *T=tabella, x=elemento da inserire*

Costo Inserimento

$$\Theta(1)$$

CHAINED-HASH-SEARCH(T, k)

```
1   return LIST-SEARCH(T[h(x.key)], k)
```

Parametri: *T=tabella, k=chiave da cercare*
Return: *puntatore all'elemento con la chiave richiesta o* `NIL` *se non presente*

Costo Ricerca

$$\Theta(|T(h(k))|)$$

CHAINED-HASH-DELETE(T, x)

```
1   LIST-DELETE(T[h(x.key)], x)
```

Parametri: *T=tabella, x=elemento da eliminare*

Costo Eliminazione

$$\Theta(1)$$

Osservazione: Lunghezza Liste Media

Quando utilizziamo la tecnica di chaining, la **lunghezza media delle liste** ad ogni indice corrisponde al load factor

$$|T[h(k)]| = \Theta(\alpha)$$

Questo ci permette di considerare il costo computazionale della **ricerca nel caso medio** utilizzando una **funzione hash uniforme**

$$\Theta(1+\alpha)$$