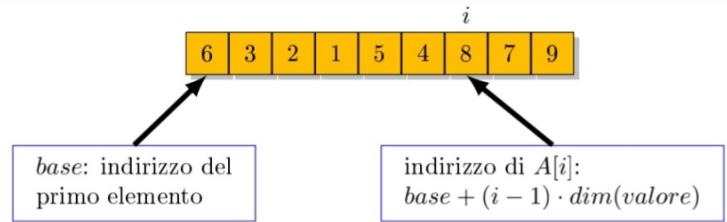


ARRAY

Calcolo dell'indirizzo di un elemento : $O(1)$

\Rightarrow Accesso diretto $\rightarrow O(1)$



ARRAY STATICO

```
ARRAYINSERT( $A, k$ )
if  $A.N \neq A.M$  then
   $A.N \leftarrow A.N + 1$ 
   $A[N] \leftarrow k$ 
  return  $k$ 
else
  return nil
end if
```

$O(1)$

```
ARRAYDELETE( $A, k$ )
for  $i \leftarrow 1$  to  $A.N$  do
  if  $A[i] == k$  then
     $A.N \leftarrow A.N - 1$ 
    for  $j \leftarrow i$  to  $A.N$  do
       $A[j] \leftarrow A[j + 1]$ 
    end for
    return  $k$ 
  end if
end for
return nil
```

$O(N)$

```
ARRAYSEARCH( $A, k$ )
for  $i \leftarrow 1$  to  $A.N$  do
  if  $A[i] == k$  then
    return  $k$ 
  end if
end for
return nil
```

$O(N)$

Se l'array e' ordinato : Inserimento $O(N)$

Cancellazione $O(N)$

Ricerca $O(\log N)$

ARRAY DINAMICO

```
ARRAYEXTEND( $A, n$ )
 $B \leftarrow$  un array con  $A.M + n$  elementi
 $B.M \leftarrow A.M + n$ 
 $B.N \leftarrow A.N$ 
for  $i \leftarrow 1$  to  $A.N$  do
     $B[i] \leftarrow A[i]$ 
end for
return  $B$ 
```

$O(N)$

→ Allocazione + copia elementi

① DYNARRAYINSERT1(A, k)

```
if  $A.N == A.M$  then
     $A \leftarrow$  ARRAYEXTEND( $A, 1$ )
end if
ARRAYINSERT( $A, k$ )
```

Dipende dallo stato dell'array

→ Array pieno → $O(N)$

→ Array non pieno → $O(1)$

aumento di 1 alla volta → SCONVENIENTE !

② DYNARRAYINSERT2(A, k)

```
if  $A.N == A.M$  then
     $A \leftarrow$  ARRAYEXTEND( $A, A.M$ )
end if
ARRAYINSERT( $A, k$ )
```

raddoppia la dimensione se A e' pieno

DYNARRAYDELETE2(A, k)

ARRAYDELETE(A, k)

if $A.N \leq 1/4 \cdot A.M$ **then**

$B \leftarrow$ un array di dimensione $A.M/2$

$B.M \leftarrow A.M/2$

$B.N \leftarrow A.N$

for $i \leftarrow 1$ to $A.N$ **do**

$B[i] \leftarrow A[i]$

end for

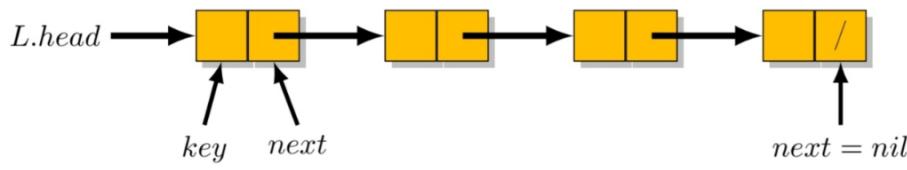
$A \leftarrow B$

end if

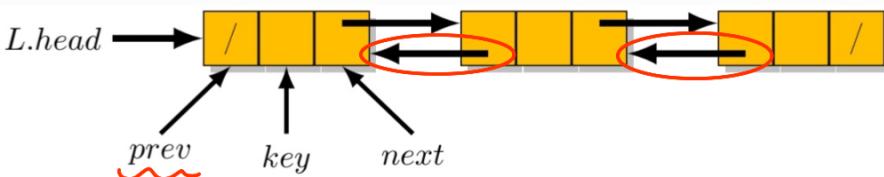
se il numero di elementi
e' $\frac{1}{4}$ della dimensione :
dimezzo

LISTE

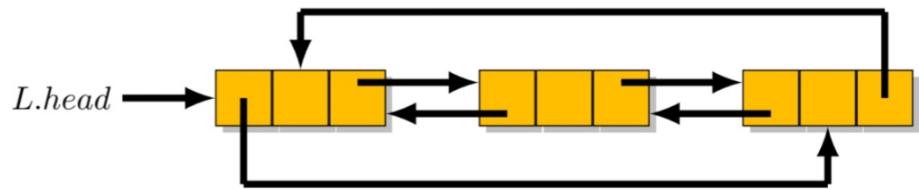
Struttura lineare in cui l'ordine degli elementi è determinato dai puntatori all'elemento successivo.



Lista doppiaamente concatenata:



Lista circolare:



OPERAZIONI PER LISTA DOPPIAMENTE CONCATENATA E
NON ORDINATA

```
LISTSEARCH( $L, k$ )      ▷ Ricerca della chiave  $k$ .  
 $x \leftarrow L.\text{head}$   
while  $x \neq \text{nil}$  and  $x.\text{key} \neq k$  do  
     $x \leftarrow x.\text{next}$   
end while  
return  $x$ 
```

$O(N)$

```
LISTINSERT( $L, x$ )      ▷ Inserimento "in testa".  
 $x.\text{next} \leftarrow L.\text{head}$   
if  $L.\text{head} \neq \text{nil}$  then  
     $L.\text{head}.\text{prev} \leftarrow x$   
end if  
 $L.\text{head} \leftarrow x$   
 $x.\text{prev} \leftarrow \text{nil}$ 
```

$O(1)$

```

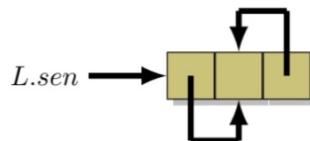
LISTDELETE( $L, x$ )      ▷ Rimozione dell'elemento puntato da  $x$ 
if  $x.prev \neq nil$  then
     $x.prev.next \leftarrow x.next$ 
else
     $L.head \leftarrow x.next$ 
end if
if  $x.next \neq nil$  then
     $x.next.prev \leftarrow x.prev$ 
end if

```

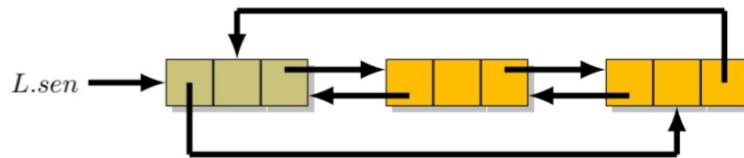
non serve
la ricerca
 $O(1)$

Possiamo aggiungere una **SENTINELLA** per rendere più omogenei gli elementi della lista ed evitare i controlli delle condizioni "in testa" e "in coda" → meno macchinoso.

La lista circolare vuota diventa :



Non Vuota :



OPERAZIONI PER LISTA DOPPIAMENTE CONCATENATA E NON ORDINATA CON SENTINELLA

```

LISTDELETESEN( $L, x$ )
l'elemento).
 $x.prev.next \leftarrow x.next$ 
 $x.next.prev \leftarrow x.prev$ 

```

▷ **Rimozione dell'elemento puntato da x**

$O(1)$

```

LISTSEARCHSEN( $L, k$ )      ▷ Ricerca della chiave  $k$ .
 $x \leftarrow L.sen.next$ 
while  $x \neq L.sen$  and  $x.key \neq k$  do
     $x \leftarrow x.next$ 
end while
return  $x$ 

```

$O(N)$

```

LISTINSERTSEN( $L, x$ )
 $x.next \leftarrow L.sen.next$ 
 $L.sen.next.prev \leftarrow x$ 
 $L.sen.next \leftarrow x$ 
 $x.prev \leftarrow L.sen$ 

```

▷ **Inserimento "in testa".**

$O(1)$

OPERAZIONI PER LISTA CONCATENATA CIRCOLARE E NON ORDINATA CON SENTINELLA

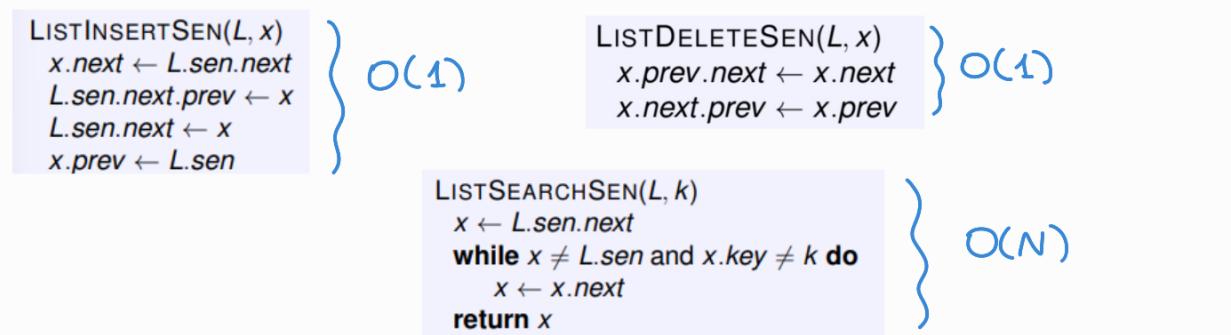


TABELLE HASH

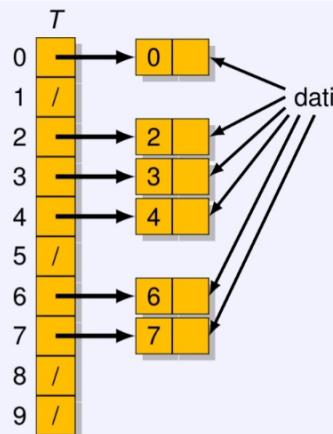
forniscono solo le operazioni di base ma ognuna e' $O(1)$

TAVOLE A INDIRIZZAMENTO DIRETTO

$U = \{0, 1, \dots, m-1\}$ universo delle chiavi

T : array di dimensione m in cui ogni posizione corrisponde a una chiave

universo delle chiavi:
 $U = \{0, 1, 2, \dots, 9\}$
insieme delle chiavi:
 $S = \{0, 2, 3, 4, 6, 7\}$



TABLEINSERT(T, x)
 $T[x.key] \leftarrow x$

TABLEDELETE(T, x)
 $T[x.key] \leftarrow nil$

TABLESEARCH(k)
return $T[k]$

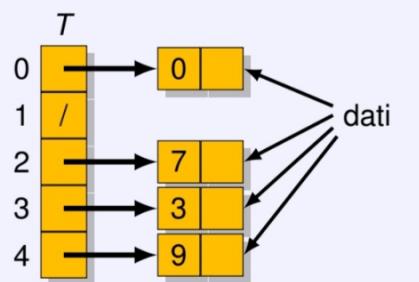
Su universi grandi e' POCO EFFICIENTE IN TERMINI DI SPAZIO

TAVOLE HASH

- Tabella T di dimensione $m \ll |U|$
- Posizione della chiave k determinata tramite una funzione hash

$$h: U \rightarrow \{0, 1, \dots, m-1\}$$

universo delle chiavi:
 $U = \{0, 1, 2, \dots, 9\}$
 insieme delle chiavi:
 $S = \{0, 3, 7, 9\}$
 funzione hash: $h(k) = k \bmod 5$
 $h(k)$ è il valore hash della chiave k



- l'indirizzamento non è più diretto
 - meno spazio utilizzato
 - $m \ll |U| \Rightarrow$ POSSONO ESSERCI DELLE COLLISIONI
 - più chiavi condividono la stessa posizione
- ⇒ Serve una buona funzione hash:
- posiziona le chiavi in modo apparentemente casuale e uniforme;
 - meno rischio di collisioni

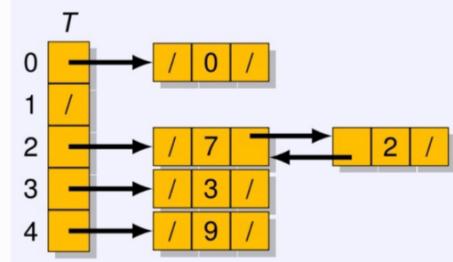
Hash perfetto : non crea mai collisioni → funzione iniettiva
 ↳ rappresentabile solo se l'insieme rappresentato non è dinamico

Per risolvere le collisioni : TAVOLE HASH CON CONCATENAMENTO

TAVOLE HASH CON CONCATENAMENTO

→ Collisioni concatenate in una lista

universo delle chiavi:
 $U = \{0, 1, 2, \dots, 9\}$
 insieme delle chiavi:
 $S = \{0, 2, 3, 7, 9\}$
 funzione hash: $h(k) = k \bmod 5$



OPERAZIONI :

HASHINSERT(T, x)
 $L \leftarrow T[h(x.key)]$
 LISTINSERT(L, x)

$O(1)$

HASHSEARCH(T, k)
 $L \leftarrow T[h(k)]$
 return LISTSEARCH(L, k)

$T[h(k)]$

HASHDELETE(T, x)
 $L \leftarrow T[h(x.key)]$
 LISTDELETE(L, x)

$O(1)$

dipende dalla lunghezza
 della lista



Nel caso peggiore : $\Theta(N)$

Nel caso migliore : $O(1)$

Nel caso medio : dipende dalla funzione hash.

Se gode di uniformità semplice
 $O(1)$

FUNZIONI HASH

- metodo della divisione : $h(k) = k \bmod m$
 - molto veloce
 - bisogna scegliere m bene → numero primo non vicino ad una potenza di 2
- metodo della moltiplicazione : $0 < A < 1$ $h(k) = \lfloor m(Ak \bmod 1) \rfloor$
 - $m = \text{potenza di 2}$
 - scelta ottimale di A : dipende dai dati

INDIRIZZAMENTO APERTO

- tutti gli elementi sono memorizzati nella tavola T
- l'elemento con chiave k viene inserito in posizione $h(k)$ se è libera;

Se non lo è, si cerca una posizione libera secondo uno SCHEMA DI ISPEZIONE

più semplice:

ISPEZIONE LINEARE → vado nella prima cella libera dopo $h(k)$

Esempio con ispezione lineare:

- universo delle chiavi: $U = \{0, 1, 2, \dots, 99\}$
- sequenza di inserimento: 88, 12, 2, 22, 33
- funzione hash: $h(k) = k \bmod 10$

T	/
0	/
1	/
2	/
3	/
4	/
5	/
6	/
7	/
8	/
9	/

T	/
0	/
1	/
2	/
3	/
4	/
5	/
6	/
7	/
8	88
9	/

$$88 \bmod 10 = 8 \\ \text{OK}$$

T	/
0	/
1	/
2	12
3	/
4	/
5	/
6	/
7	/
8	88
9	/

$$12 \bmod 10 = 2 \\ \text{OK}$$

T	/
0	/
1	/
2	12
3	2
4	/
5	/
6	/
7	/
8	88
9	/

$$2 \bmod 10 = 2 \\ \text{GIÀ OCCUPATA} \\ \Rightarrow 3$$

T	/
0	/
1	/
2	12
3	2
4	22
5	/
6	/
7	/
8	88
9	/

$$22 \bmod 10 = 2 \\ \text{GIÀ OCCUPATA} \\ \Rightarrow 3 \text{ OCCUPATA} \\ \Rightarrow 4$$

T	/
0	/
1	/
2	12
3	2
4	22
5	33
6	/
7	/
8	88
9	/

$$33 \bmod 10 = 3 \\ \text{GIÀ OCCUPATA} \\ \Rightarrow 4 \text{ GIÀ OCCUPATA} \\ \Rightarrow 5$$

Ispezione lineare se

$$h(k, i) = (f_i(k) + i) \bmod m$$

mi sposto
in avanti
per garantire
la circolarità

funzione "normale"
 (considera solo la chiave)

OPERAZIONI CON INDIRIZZAMENTO APERTO:

HASHINSERT(T, x)

$i \leftarrow 0$

while $i < m$ **do**

$$j \leftarrow h(x.key, i)$$

if $T[j] = \text{nil}$ **then**

$T[j] \leftarrow x$

return j

end if

$i \leftarrow i + 1$

end while

return nil

HASHSEARCH(T, k)

$i \leftarrow 0$

while $i < m$ **do**

$$j \leftarrow h(k, i)$$

if $T[j] = \text{nil}$ **then** **return** nil) K non c'e'

end if

if $T[j].key = k$ **then**
return $T[j]$

end if

$i \leftarrow i + 1$) posizione occupata
NON da $k \rightarrow$ vado
avanti

and while

return *nil*

L'indirizzamento aperto si usa quando non c'e' necessita di cancellare !

Ispettione lineare → crea ADDENSAMENTO PRIMARIO

= addensamento locale attorno alla chiave

⇒ ISPEZIONE QUADRATICA

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

↳ crea ADDENSAMENTO SECONDARIO

$$\Rightarrow \text{DOPPIO HASHING} : h(k, i) = (h_1(k) + i h_2(k)) \bmod m$$