

Realizzazione di puntatori ed oggetti

Possiamo realizzare puntatori ed oggetti anche in linguaggi che non li forniscono

Rappresentazione di oggetti con array multipli

Possiamo rappresentare un insieme di oggetti che abbiano gli stessi campi usando un array per ogni campo

Lista concatenata:

x , $key[x]$, $next[x]$, $prev[x]$

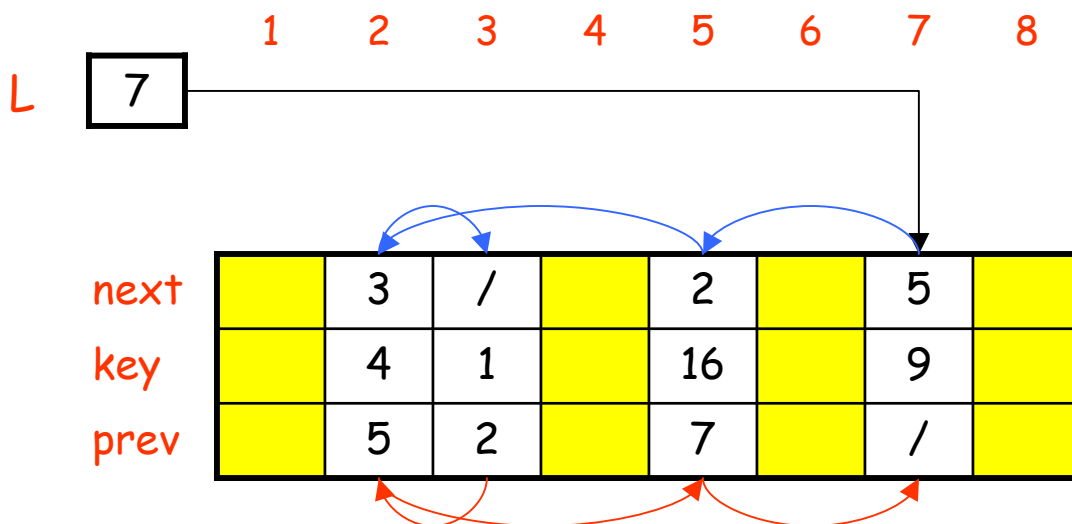
x = puntatore all'oggetto = indice comune per key , $next$ e $prev$

Key = array contiene le chiavi

$next$ = contiene gli indici (i puntatori) agli oggetti successivi

$prev$ = contiene gli indici (i puntatori) agli oggetti precedenti

L = variabile che contiene l'indice L della testa della lista



Rappresentazione di oggetti con un array singolo

In molti linguaggi un oggetto occupa un insieme di locazioni contigue della memoria del calcolatore. Un puntatore e' semplicemente l'indirizzo della prima locazione occupata dall'oggetto. Le altre locazioni possono essere raggiunte aggiungendo uno spostamento al puntatore

Possiamo usare questa strategia per realizzare oggetti utilizzando un array singolo in ambienti di programmazione che non forniscono tipi di dato puntatore

Lista concatenata = insieme di oggetti a 3 campi (key, next, prev)

Un oggetto occupa un sotto-array contiguo $A[j, j+1, \dots, j+n-1]$

j = puntatore all'oggetto

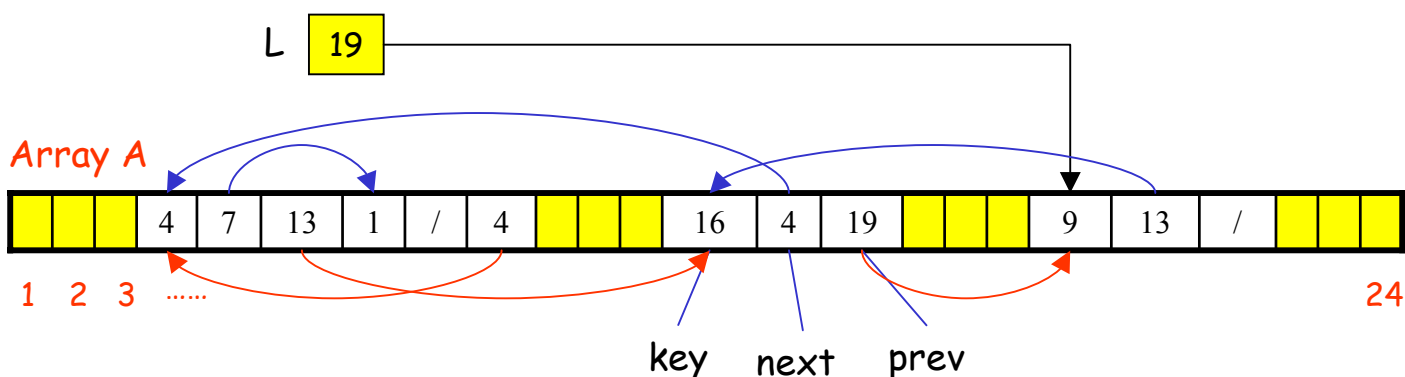
n = numero di campi dell'oggetto

$A[j]$ = chiave dell'oggetto

$A[j+1]$ = indice (puntatore) dell'oggetto successivo

$A[j+2]$ = indice (puntatore) dell'oggetto precedente

L = variabile che contiene l'indice L della testa della lista



Allocazione e deallocazione di oggetti

Per inserire un oggetto in un insieme dinamico rappresentato con una lista bidirezionale, si deve allocare un puntatore ad un oggetto inutilizzato nella rappresentazione della lista concatenata → **gestione della memoria**

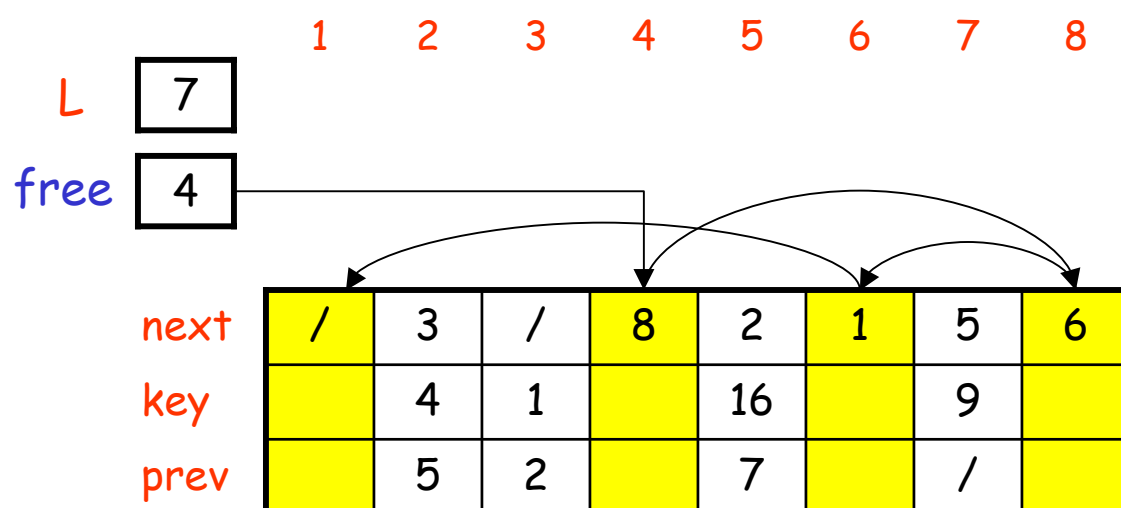
Analizziamo il problema di allocare e deallocare oggetti considerando l'esempio della lista bidirezionale rappresentata con array multipli.

m = lunghezza array

n = numero di oggetti contenuti nell'insieme

Supponiamo che, ad un certo istante, sia $n < m$. → $(m-n)$ oggetti liberi

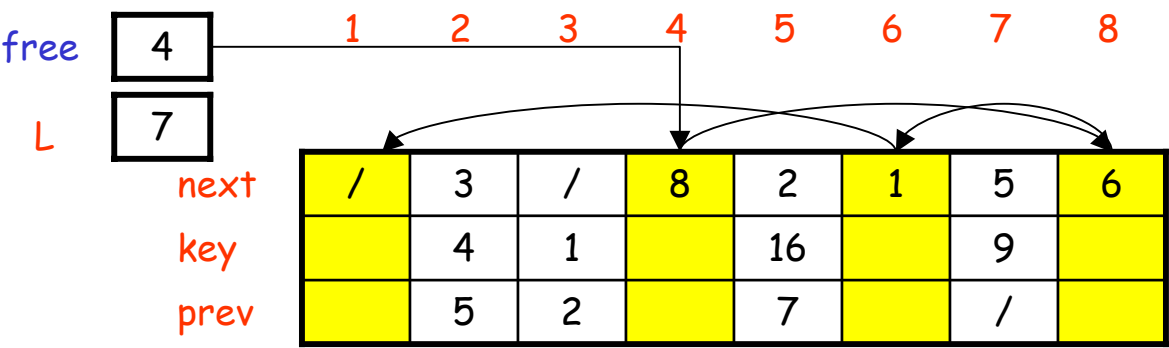
Manteniamo gli oggetti liberi in una singola lista concatenata, che si chiama **lista libera**. La lista libera usa solo il puntatore next



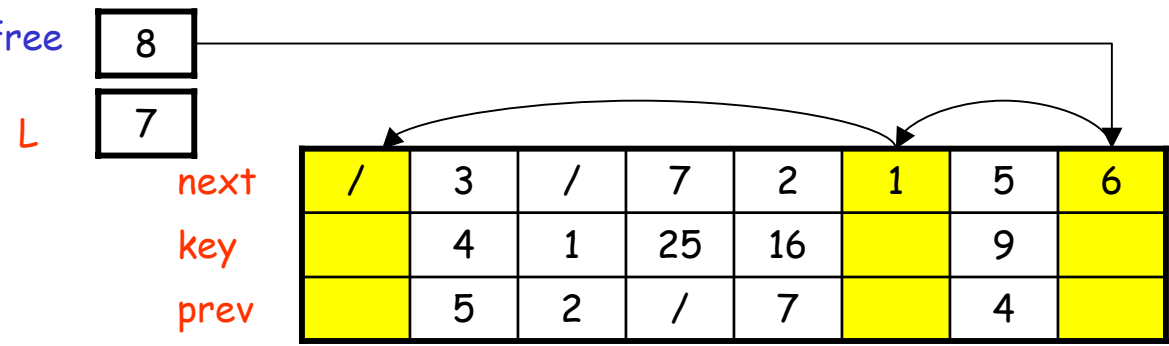
Procedure di allocazione e deallocazione

```
Allocate-Object( )  
if free = NIL  
  then error "fine dello spazio libero"  
else x ← free  
  free ← next[x]  
  return x
```

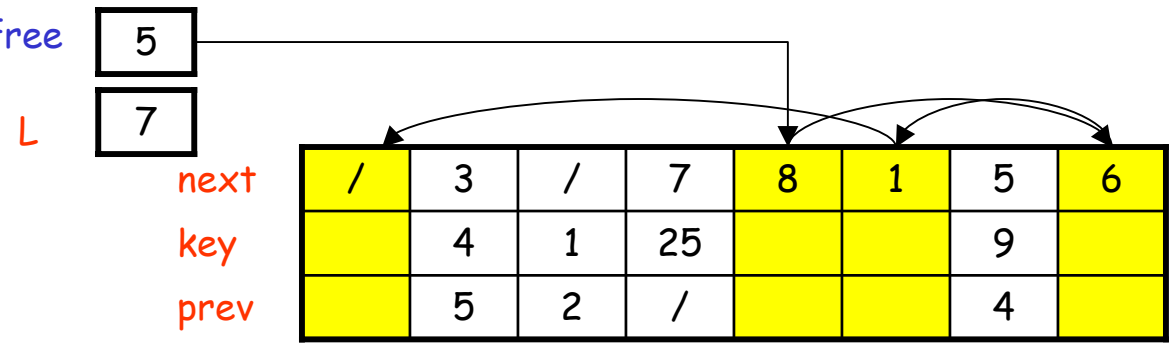
```
Free-Object(x )  
next[x]←free  
free ← x
```



```
Allocate-Object( ) List-Insert(L,4)
```



```
List-Delete(L,5) Free-Object(5)
```



Rappresentazione di alberi radicati

I metodi descritti precedentemente si applicano a tutti gli insiemi dinamici omogenei

Albero binario

insieme di oggetti (nodi) caratterizzato da quattro campi:

$K[x]$ (chiave),

$P[x]$ (puntatore al parente),

$Left[x]$ (puntatore al figlio sinistro),

$Right[x]$ (puntatore al figlio destro).

$Root[T] \rightarrow$ Puntatore alla radice dell'albero

$X =$ puntatore nodo $Left[x] = NIL$ (il nodo non ha figlio sinistro)

$Right[x] = NIL$ (il nodo non ha figlio destro)

Albero radicato con numero limitato di figli -

n = numero massimo di figli di un nodo.

insieme di oggetti (nodi) caratterizzato da $n+2$ campi:

$K[x]$ (chiave),

$P[x]$ (puntatore al parente),

$Child1[x]$ (puntatore al primo figlio),

$Child2[x]$ (puntatore al secondo figlio),

.....

.....

$child_n[x]$ (puntatore allo n -esimo figlio).

Albero radicato con un numero illimitato di figli

Rappresentazione figlio-sinistro fratello-destro

Ogni nodo x contiene:

$K[x]$ = chiave,

$\text{Left-child}[x]$ = puntatore al figlio piu' a sinistra di x ,

$\text{Right-sibling}[x]$ = punta al fratello immediatamente alla destra di x

Se il nodo x non ha figli, $\text{left-child}[x] = \text{NIL}$ e se il nodo x e' il figlio piu' a destra di suo padre allora $\text{Right-sibling}[x] = \text{NIL}$.