

# Algoritmi e Strutture di Dati

Liste implementate tramite array

*m.patrignani*

090-liste-tramite-array-02 copyright ©2018 patrignani@dia.uniroma3.it

## Nota di copyright

- queste slides sono protette dalle leggi sul copyright
- il titolo ed il copyright relativi alle slides (inclusi, ma non limitatamente, immagini, foto, animazioni, video, audio, musica e testo) sono di proprietà degli autori indicati sulla prima pagina
- le slides possono essere riprodotte ed utilizzate liberamente, non a fini di lucro, da università e scuole pubbliche e da istituti pubblici di ricerca
- ogni altro uso o riproduzione è vietata, se non esplicitamente autorizzata per iscritto, a priori, da parte degli autori
- gli autori non si assumono nessuna responsabilità per il contenuto delle slides, che sono comunque soggette a cambiamento
- questa nota di copyright non deve essere mai rimossa e deve essere riportata anche in casi di uso parziale

090-liste-tramite-array-02 copyright ©2018 patrignani@dia.uniroma3.it

## Sommario

- Il tipo astratto di dato lista
  - tipologie di liste e strategie di realizzazione
- Realizzazione delle liste con array
  - uso di tre array
  - uso di un solo array
  - liste disomogenee

090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Liste

- Le liste sono strutture di dati in cui gli oggetti sono disposti in una sequenza lineare
  - si assume che l'utente voglia scorrere gli elementi tramite un iteratore
- Tipo astratto: lista di interi
  - domini
    - il dominio di interesse è l'insieme delle liste  $L$  di interi
    - dominio di supporto: gli iteratori  $I$  che identificano le posizioni
    - dominio di supporto: gli interi  $Z = \{0, 1, -1, 2, -1, \dots\}$
    - dominio di supporto: i booleani  $B = \{\text{true}, \text{false}\}$

090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Realizzazione di una lista

- Costanti
  - la lista vuota viene realizzata tramite la funzione `NEW_LIST(maxsize)`
    - ritorna il riferimento ad una lista vuota che può contenere al massimo `maxsize` elementi
  - l'iteratore non valido è solitamente uno specifico valore dell'iteratore
- Operazioni di aggiornamento
  - l'inserimento in testa alla lista `INSERT: L × Z → L` viene realizzato tramite la funzione `INSERT(l,x)`
  - l'inserimento in coda `ADD: L × Z → L` viene realizzato tramite la funzione `ADD(l,x)`
  - l'eliminazione di un elemento a partire dal suo iteratore `DELETE: L × I → L` viene realizzata tramite la funzione `DELETE(l,i)`
  - la ricerca e l'eliminazione di un elemento `DELETE: L × Z → L` viene realizzata tramite la funzione `DELETE(l,x)`

090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Operazioni possibili sulle liste

- Altre operazioni di aggiornamento
  - l'inserimento prima di una posizione specifica `INSERT_BEFORE: L × I × Z → L` viene realizzato tramite la funzione `INSERT_BEFORE(l,x,i)`
  - l'inserimento dopo una posizione specifica `ADD_AFTER: L × I × Z → L` viene realizzato tramite la funzione `ADD_AFTER(l,x,i)`
  - lo svuotamento della lista `EMPTY: L → L` viene realizzato tramite la funzione `EMPTY(l)`

090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Operazioni possibili sulle liste

### • Operazioni di consultazione

- l'iteratore del 1° elemento della lista HEAD:  $L \rightarrow I$  si ottiene tramite la funzione HEAD(1)
  - ritorna l'iteratore non valido se la lista è vuota
- l'iteratore successivo all'iteratore corrente NEXT:  $L \times I \rightarrow I$  si ottiene tramite la funzione NEXT(1,i)
  - ritorna l'iteratore non valido se i è l'iteratore dell'ultimo elemento
- l'iteratore precedente all'iteratore corrente PREV:  $L \times I \rightarrow I$  si ottiene tramite la funzione PREV(1,i)
  - ritorna l'iteratore non valido se i è l'iteratore del primo elemento
- l'intero associato alla posizione specificata da un iteratore INFO:  $L \times I \rightarrow Z$  si ottiene con la funzione INFO(1,i)

090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Operazioni possibili sulle liste

### • Altre operazioni di consultazione

- la ricerca della posizione di un elemento SEARCH:  $L \times Z \rightarrow I$  si ottiene tramite la funzione SEARCH(1,k)
  - ritorna l'iteratore dell'elemento con chiave k nella lista 1 oppure l'iteratore non valido
- l'iteratore associato all'ultimo elemento della lista LAST:  $L \rightarrow I$  si ottiene tramite la funzione LAST(1)
  - ritorna l'iteratore non valido se la lista è vuota
- la verifica se una lista è vuota IS\_EMPTY:  $L \rightarrow B$  è realizzata dalla funzione IS\_EMPTY(1)
- il numero degli elementi in lista SIZE:  $L \rightarrow Z$  si ottiene tramite la funzione SIZE(1)

090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Strategie di realizzazione delle liste

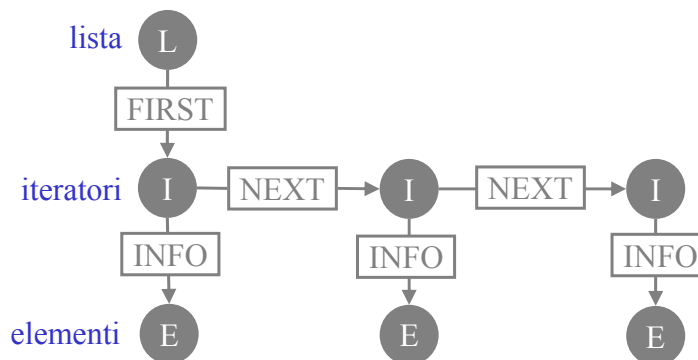
- Dipendentemente dal tipo di operazioni che è necessario compiere in maniera efficiente sulla lista esistono diverse strategie implementative:
  - lista concatenata
    - consente lo scorrimento efficiente della lista in avanti ma non consente un efficiente scorrimento all'indietro
  - lista doppiamente concatenata
    - supporta in maniera efficiente lo scorrimento in avanti e indietro
  - accesso agli estremi
    - consente un veloce accesso sia al primo che all'ultimo elemento della lista

090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Lista semplicemente concatenata

- La struttura di dati supporta il passaggio diretto da un iteratore all'iteratore successivo
  - si vogliono le operazioni FIRST e NEXT in  $\Theta(1)$

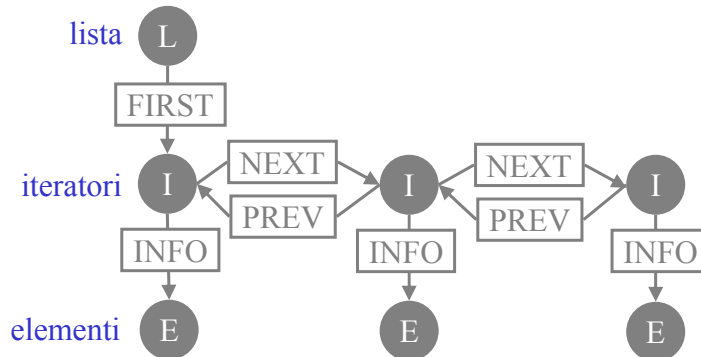


090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Lista doppiamente concatenata

- La struttura di dati supporta il passaggio diretto da un iteratore al successivo e al precedente
  - si vogliono le operazioni FIRST, NEXT e PREV in  $\Theta(1)$

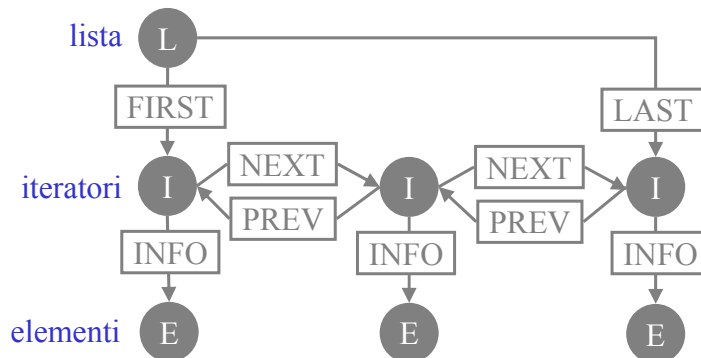


090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Liste con accesso agli estremi

- La struttura di dati supporta l'accesso diretto al primo e all'ultimo iteratore della lista
  - si vogliono le operazioni FIRST e LAST in  $\Theta(1)$

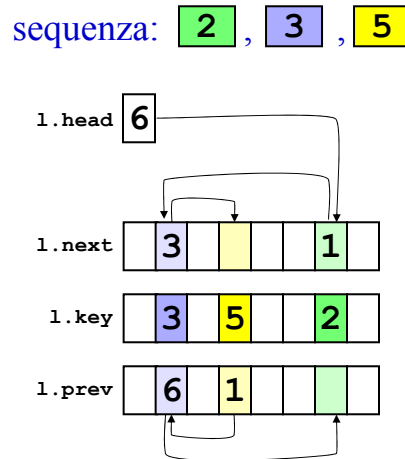


090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Realizzazione di una lista con array

- Gli elementi della lista sono memorizzati in un array `l.key`
- L'array `l.next` contiene l'indice dell'elemento che segue
- L'array `l.prev` contiene l'indice dell'elemento che segue

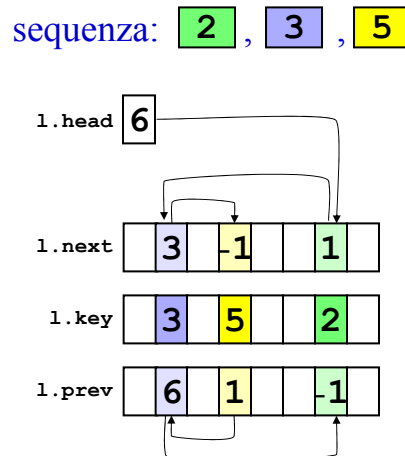


090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Realizzazione di una lista con array

- L'iteratore della lista è un intero
  - è l'indice della posizione dell'elemento corrispondente
- L'iteratore non valido è rappresentato dal valore -1



090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Più liste con gli stessi array?

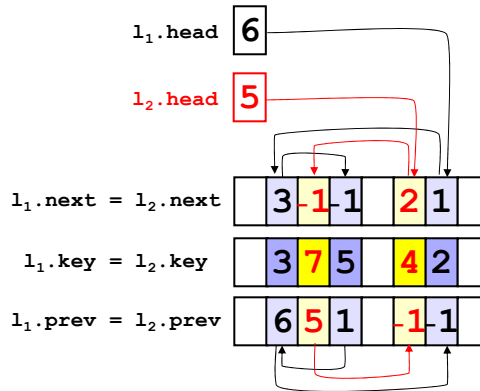
- Gli array  
`l.next`, `l.key`  
`l.prev`

possono essere  
condivisi da due  
o più liste

- le liste non  
interferiscono,  
perché utilizzano  
posizioni diverse  
degli array

sequenza 1: 2, 3, 5

sequenza 2: 4, 7

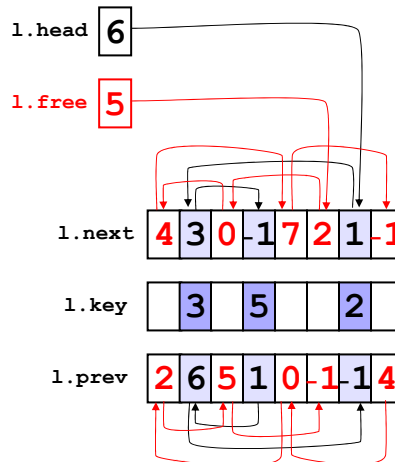


090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Uso della lista libera

- Per inserire un nuovo elemento in lista occorre sapere quali posizioni degli array sono ancora libere
- Tutte le posizioni libere possono essere memorizzate in una seconda lista `l.free`
  - un inserimento di un elemento in `l` è un trasferimento di una posizione dalla lista `l.free` alla lista `l.head`
  - una cancellazione da `l` è un trasferimento di una posizione dalla lista `l.head` alla lista `l.free`



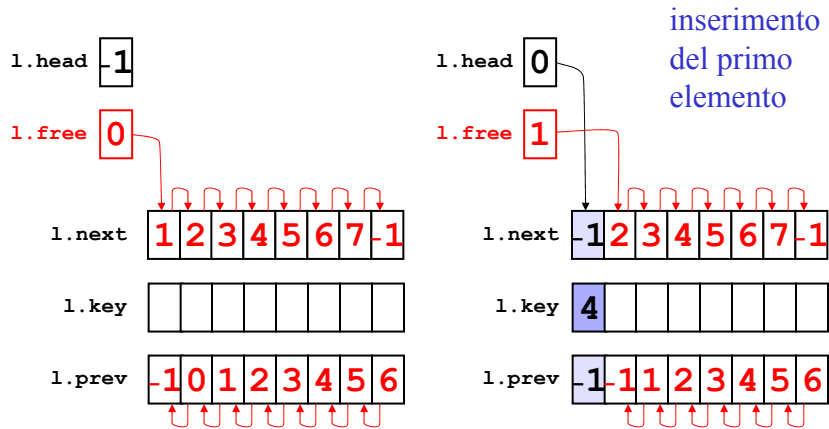
090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it



## Configurazione iniziale: lista vuota

- Quando la lista `l` è vuota tutte le posizioni sono assegnate alla lista `l.free`



090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Creazione di una lista vuota di interi

- Procedura per inizializzare una lista vuota di interi

```
NEW_LIST(maxsize)
```

```
1. ▷ creo un nuovo oggetto l con:
2. ▷   l.next, l.key, l.prev array di maxsize interi
3. ▷   l.head, l.free interi
4. for i = 0 to maxsize-1
5.   l.next[i] = i+1
6.   l.prev[i] = i-1
7. l.next[maxsize-1] = -1
8. l.head = -1
9. l.free = 0
10. return l
```

090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it

## Gestione della lista `l.free`

- Procedura di servizio per ottenere una posizione libera dalla lista `l.free`

```

ALLOCATE-COLUMN(l)
1. if l.free == -1
2.   error("overflow")
3. else
4.   i = l.free    ▷ i è l'indice della nuova posizione
5.   l.free = l.next[l.free]
6.   if l.free != -1
7.     l.prev[l.free] = -1
8. return i

```

090-liste-tramite-array-02    copyright ©2018 patrignani@dia.uniroma3.it

## Gestione della lista `l.free`

- Procedura di servizio per restituire una posizione alla lista `l.free`

```

FREE-COLUMN(l,i)
1. l.prev[i] = -1
2. l.next[i] = l.free
3. if l.free != -1
4.   l.prev[l.free] = i
5. l.free = i

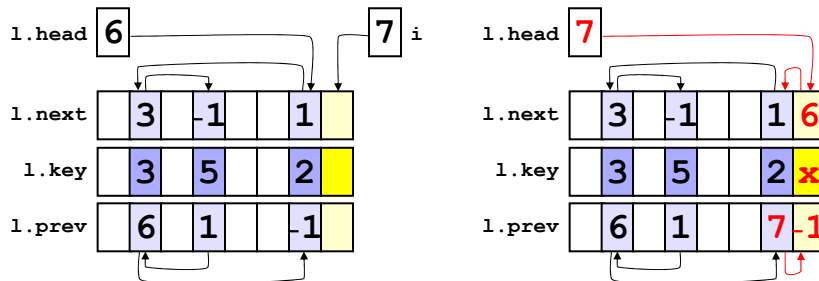
```

090-liste-tramite-array-02    copyright ©2018 patrignani@dia.uniroma3.it

## Codice di INSERT(l,x)

```

INSERT(l,x)      ▷ x è il valore da aggiungere in lista
1. i = ALLOCATE-COLUMN(l)    ▷ indice di una nuova colonna libera
2. l.key[i] = x
3. l.prev[i] = -1              ▷ i diventa primo elemento
4. l.next[i] = l.head          ▷ il resto della lista segue i
5. if l.head != -1
6.     l.prev[l.head] = i
7. l.head = i
  
```

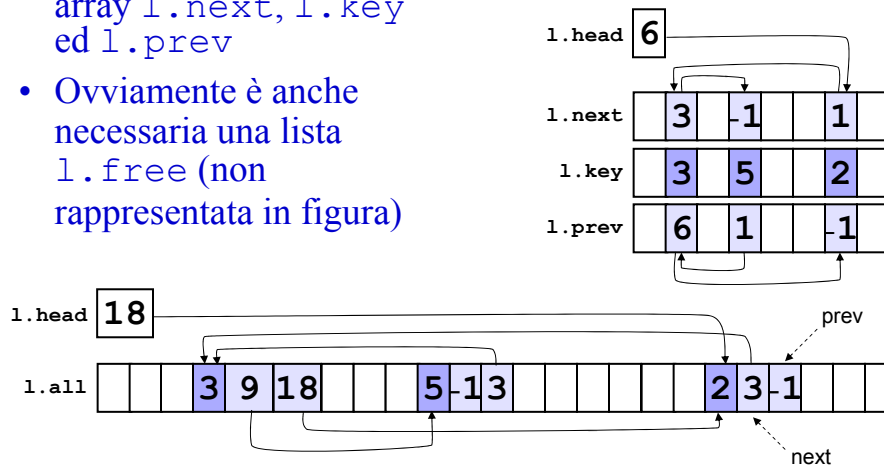


## Esercizi: liste implementate con array

1. Scrivi lo pseudocodice della procedura `SIZE(l)` che conta gli elementi della lista `l`
2. Scrivi lo pseudocodice della procedura `SEARCH(l,k)` che restituisce la posizione del primo elemento di `l` con valore della chiave `k`
3. Scrivi lo pseudocodice della procedura `DELETE(l,i)` che rimuove da `l` l'elemento in posizione `i`
4. Scrivi lo pseudocodice della procedura `DELETE(l,x)` che rimuove da `l` il primo elemento che ha valore `x`

## Rappresentazione con un solo array

- Un solo array è sufficiente a rappresentare le informazioni degli array `l.next`, `l.key` ed `l.prev`
- Ovviamente è anche necessaria una lista `l.free` (non rappresentata in figura)



090-liste-tramite-array-02 copyright ©2018 patrignani@dia.uniroma3.it

## Esercizi: liste su un solo array

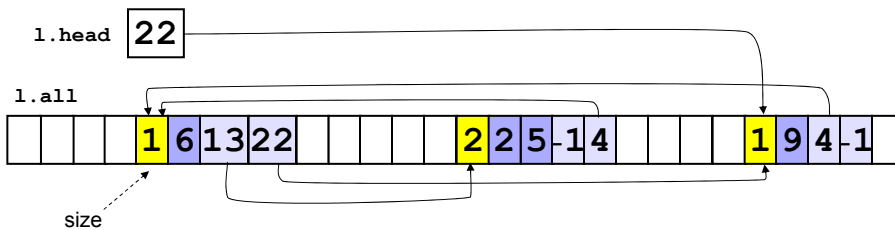
*Negli esercizi seguenti supponi che la lista `l` sia doppiamente concatenata ed implementata tramite un solo array*

- Scrivi lo pseudocodice della procedura `ALLOCATE_OBJECT(l)`
- Scrivi lo pseudocodice della procedura `FREE_OBJECT(l,i)`
- Scrivi lo pseudocodice della procedura `LOWER_FREE_POSITION(l)` che trova la posizione con indice più basso tra gli elementi della lista libera `l.free`

090-liste-tramite-array-02 copyright ©2018 patrignani@dia.uniroma3.it

## Liste con elementi eterogenei

- L'uso di un singolo array consente la gestione di liste di elementi eterogenei
  - un campo aggiuntivo specifica la dimensione di ogni elemento
  - la lista `l.free` (non rappresentata in figura) viene gestita con criteri analoghi



090-liste-tramite-array-02 copyright ©2018 patrignani@dia.uniroma3.it

## Esercizi: liste eterogenee

*negli esercizi seguenti supponi che la lista `l` sia eterogenea, doppiamente concatenata, ed implementata tramite un singolo array*

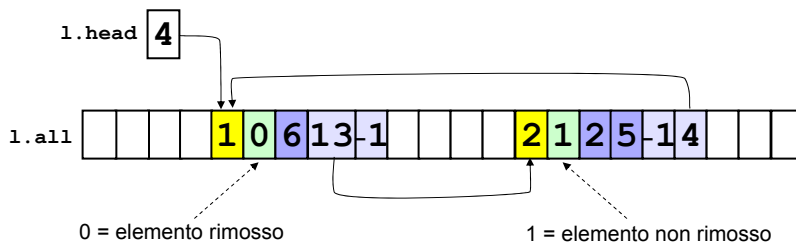
8. Scrivi lo pseudocodice della procedura `ALLOCATE-OBJECT-WITH-SIZE(l,x)` che trova nella lista libera `l.free` una posizione adatta ad ospitare un elemento di dimensione `x`
9. Scrivi lo pseudocodice della procedura `FREE-OBJECT(l,i)` che inserisce nella lista libera `l.free` l'oggetto in posizione `i`
10. Scrivi lo pseudocodice della procedura `INSERT(l,A)` che inserisce nella lista `l` un elemento di dimensione `A.length` che contiene tutti i valori dell'array `A`

090-liste-tramite-array-02 copyright ©2018 patrignani@dia.uniroma3.it

## Esercizio: cancellazione *lazy* su liste eterogenee

*Negli esercizi seguenti supponi che una lista eterogenea doppiamente concatenata ed implementata con un singolo array preveda, per ogni elemento, un valore che specifica se l'elemento è da considerarsi "rimosso" oppure no*

11. Scrivi lo pseudocodice della procedura `DELETE(l,i)` dove  $i$  è l'indice della posizione di un elemento da marcare come "rimosso"
12. Scrivi lo pseudocodice della procedura `GARBAGE-COLLECTION(l)` che trasferisce nella lista `l.free` tutti gli elementi marcati come "rimossi"



090-liste-tramite-array-02

copyright ©2018 patrignani@dia.uniroma3.it