

Algoritmi e Strutture di Dati

Implementazioni di liste con
oggetti e riferimenti

m.patrignani

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

Strutture di dati con oggetti e riferimenti

- alcuni linguaggi supportano oggetti e riferimenti
 - lo pseudocodice è uno di questi
- con oggetti e riferimenti si possono realizzare strutture di dati elementari in modo più naturale
- vedremo la realizzazione del tipo astratto di dato lista
 - pile e code possono essere rivisti come casi particolari di liste
 - due principali varianti implementative:
 - lista singolarmente concatenata
 - lista doppiamente concatenata

Operazioni su una lista di interi

<code>NEW_LIST()</code>	ritorna il riferimento ad una lista vuota
<code>HEAD(l)</code>	ritorna l'iteratore del primo elemento della lista
<code>LAST(l)</code>	ritorna l'iteratore dell'ultimo elemento della lista
<code>NEXT(l,i)</code>	ritorna l'iteratore dell'elemento che segue <code>i</code> nella lista ritorna un iteratore invalido se <code>i</code> è l'ultimo elemento
<code>PREV(l,i)</code>	ritorna l'iteratore dell'elemento che precede <code>i</code> nella lista ritorna un iteratore invalido se <code>i</code> è il primo elemento
<code>INSERT(l,n)</code>	inserisce l'elemento <code>n</code> in testa alla lista <code>l</code>
<code>INSERT_BEFORE(l,n,i)</code>	inserisce l'elemento <code>n</code> prima della posizione <code>i</code>
<code>ADD(l,n)</code>	aggiunge <code>n</code> in coda alla lista <code>l</code>
<code>ADD_AFTER(l,n,i)</code>	aggiunge l'elemento <code>n</code> dopo la posizione <code>i</code>
<code>DELETE(l,i)</code>	rimuove l'elemento in posizione <code>i</code> dalla lista <code>l</code>
<code>DELETE(l,n)</code>	rimuove l'elemento <code>n</code> dalla lista <code>l</code>
<code>EMPTY(l)</code>	vuota la lista
<code>SEARCH(l,n)</code>	ritorna l'iteratore dell'elemento <code>n</code> nella lista <code>l</code>
<code>IS_EMPTY(l)</code>	ritorna <code>true</code> se la lista è vuota, altrimenti ritorna <code>false</code>

Lista concatenata (singly linked list)

- l'iteratore `i` è un riferimento ad un nodo della lista, che è un oggetto composto dai seguenti attributi

- `i.info`

- elemento in lista del tipo opportuno
 - può essere un riferimento ad un oggetto esterno con dati satellite

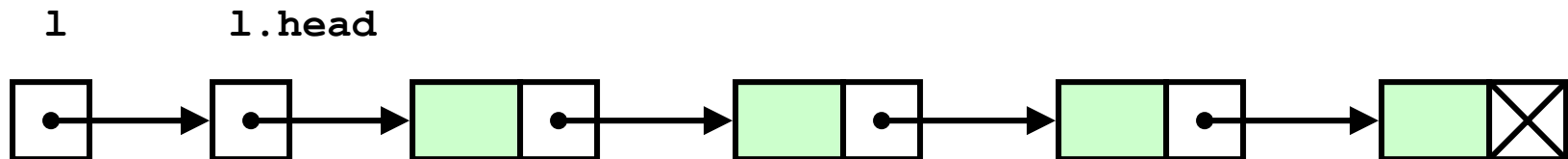
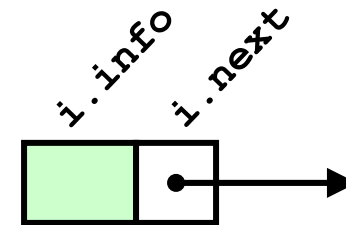
- `i.next`

- riferimento al nodo seguente o NULL

- una lista `l` ha un solo attributo

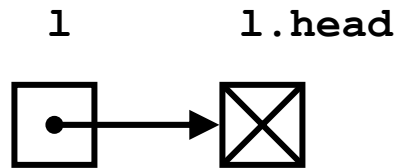
- `l.head`

- riferimento al primo nodo



Lista concatenata: lista vuota

- Quando la lista `l` è vuota `l.head` è `NULL`



- Pseudocodice delle procedure `IS_EMPTY` e `EMPTY`

```
IS_EMPTY(l)
```

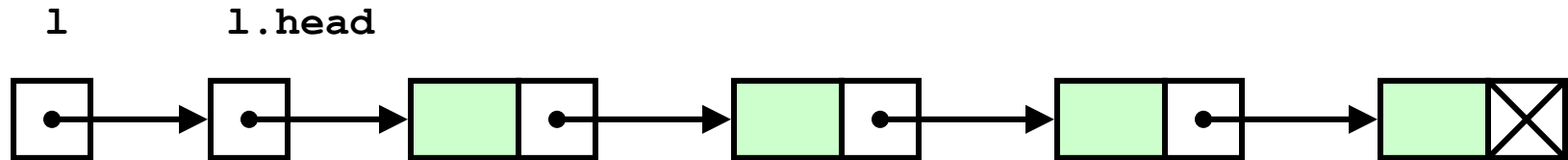
```
1. return l.head == NULL
```

```
EMPTY(l)
```

```
1. l.head = NULL      ▷ lo pseudocodice non dealloca memoria
```

Lista concatenata: first e next

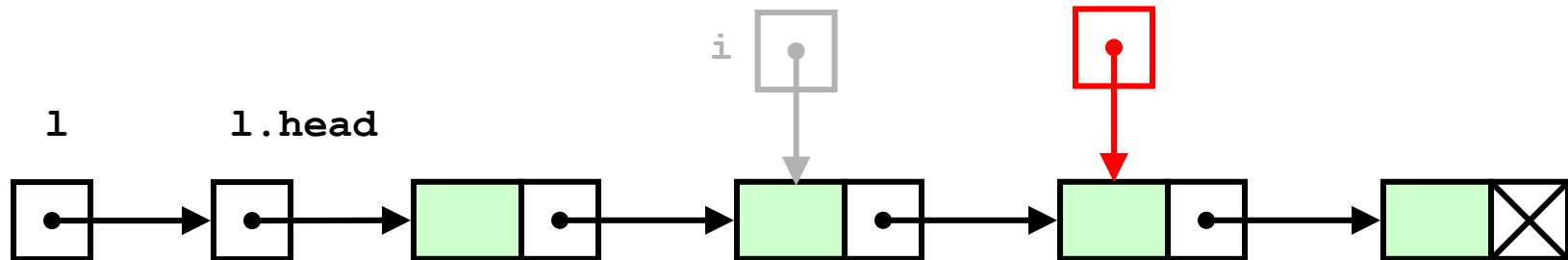
- FIRST: iteratore dell'elemento affiorante



FIRST(l)

1. **return** l.head ▷ potrebbe essere NULL

- NEXT: prossimo elemento



NEXT(l, i)

1. **return** i.next ▷ il parametro l non è utilizzato

Realizzazione di funzioni elementari

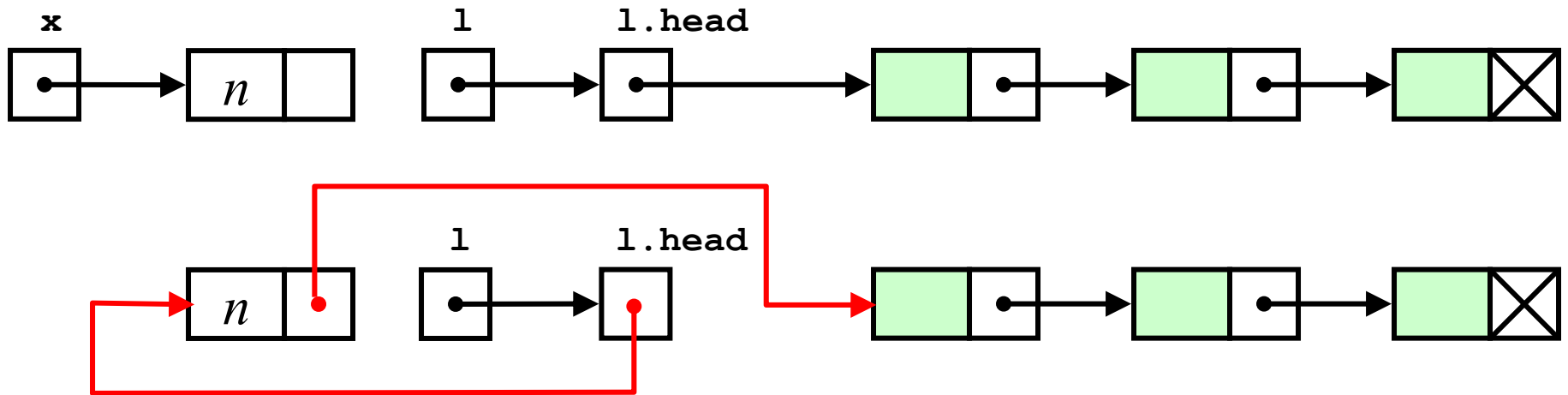
- La semplicità di alcune funzioni (come `IS_EMPTY`, `EMPTY`, `FIRST`, `NEXT`, ecc) induce a sostituirle con le istruzioni opportune direttamente nello pseudocodice
 - questo ovviamente fa perdere di generalità al codice scritto
- Esempio

```
...  
    if !IS-EMPTY(l)  
    then ...  
...  
    x = NEXT(l, x)  
...
```

```
...  
    if l.head != NULL  
    then ...  
...  
    x = x.next  
...
```


Lista concatenata: inserimento in testa

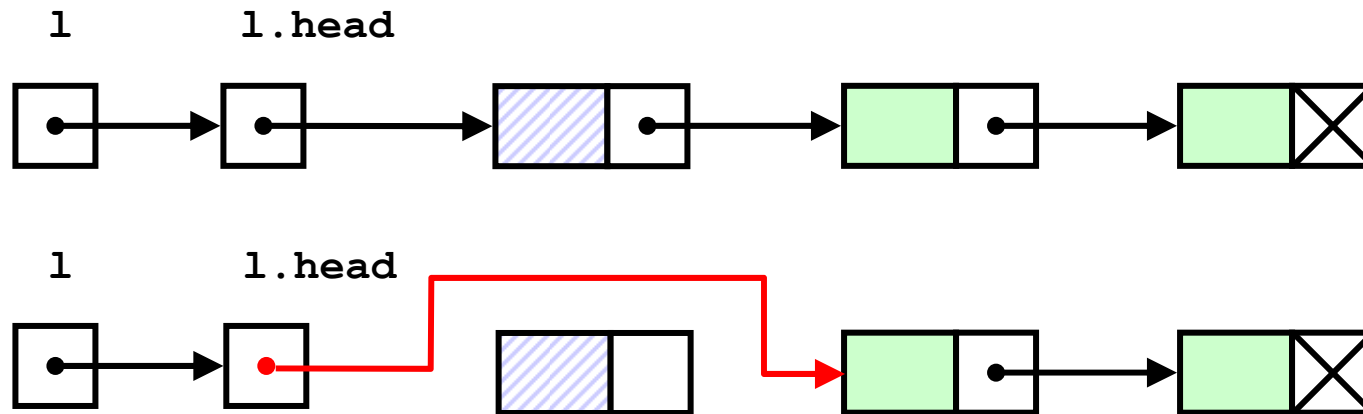
- INSERT: inserimento di n in testa alla lista



```
INSERT (l, n)    ▷ n è un intero
1. x.info = n    ▷ x è un nuovo oggetto con due campi:
2.              ▷ x.info (intero)
3.              ▷ x.next (rif. ad analogo oggetto)
4. x.next = l.head
5. l.head = x
```

Lista concatenata: cancellazione

- DELETE_FIRST: rimozione del primo nodo



```
DELETE_FIRST(1)
```

```
1. ▷ NOTA: lo pseudocodice non dealloca l'elemento
2. if l.head == NULL
3.     error("lista vuota")
4. else
5.     l.head = l.head.next
```

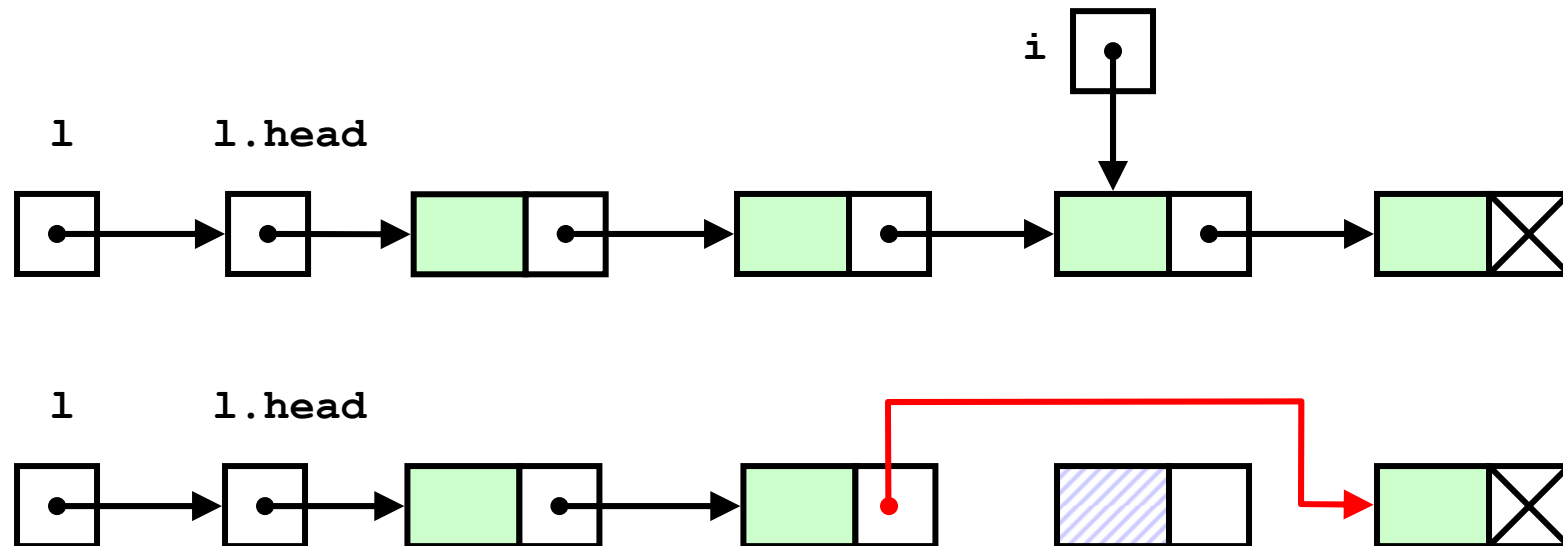
Esercizi: liste singolarmente concatenate

Esercizi sullo scorrimento delle liste

1. Scrivi lo pseudocodice della procedura `SOMMA(l)` che ritorna la somma degli elementi contenuti in una lista singolarmente concatenata di interi
2. Scrivi lo pseudocodice della procedura `MASSIMO(l)` che ritorna il valore del massimo elemento contenuto in una lista singolarmente concatenata di interi
 - assumi che la lista non sia mai vuota

Lista concatenata: cancellazione

- $\text{DELETE}(l, i)$: cancellazione del nodo i
- La cancellazione di un nodo diverso dal primo è poco efficiente in una lista singolarmente concatenata



- Occorre infatti modificare l'attributo `next` del nodo che lo precede

Esercizi: liste singolarmente concatenate

3. Scrivi lo pseudocodice della procedura $\text{SEARCH}(l, u)$ che ritorna il riferimento all'elemento i che contiene il valore intero u in una lista singolarmente concatenata di interi (oppure NULL se u non è nella lista)
 - discuti la complessità dell'algoritmo in funzione del numero n degli elementi in lista
4. Scrivi lo pseudocodice della procedura $\text{PREV}(l, i)$ che ritorna il riferimento all'elemento che precede l'elemento identificato dall'iteratore i in una lista singolarmente concatenata di interi (oppure NULL se i corrisponde al primo elemento della lista)

Esercizi: liste singolarmente concatenate

5. Scrivi lo pseudocodice dell'operazione $\text{DELETE}(l, i)$ che cancella il nodo i di una lista singolarmente concatenata
 - discuti della complessità dell'algoritmo
6. Scrivi lo pseudocodice dell'operazione $\text{DELETE}(l, u)$ che cancella il nodo che contiene il valore intero u in una lista singolarmente concatenata di interi
 - discuti della complessità dell'algoritmo

Esercizi: liste singolarmente concatenate

7. Implementa una pila di interi utilizzando oggetti e riferimenti

- devi realizzare le funzioni `NEW_STACK()`, `IS_EMPTY(p)`, `PUSH(p,u)`, e `POP(p)` facendo uso di oggetti e riferimenti

8. Implementa una coda di interi utilizzando oggetti e riferimenti

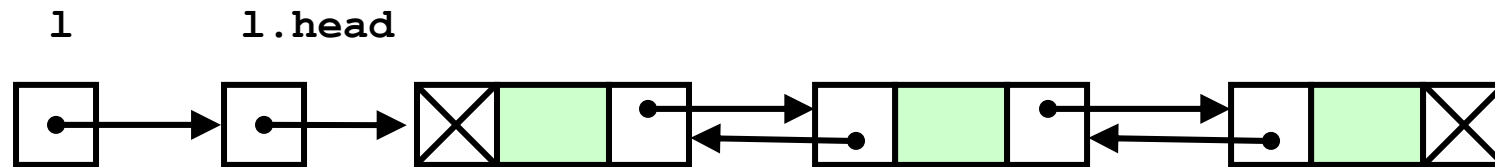
- devi realizzare le funzioni `NEW_QUEUE()`, `IS_EMPTY(c)`, `ENQUEUE(c,u)`, e `DEQUEUE(c)` facendo uso di oggetti e riferimenti

Esercizi: liste singolarmente concatenate

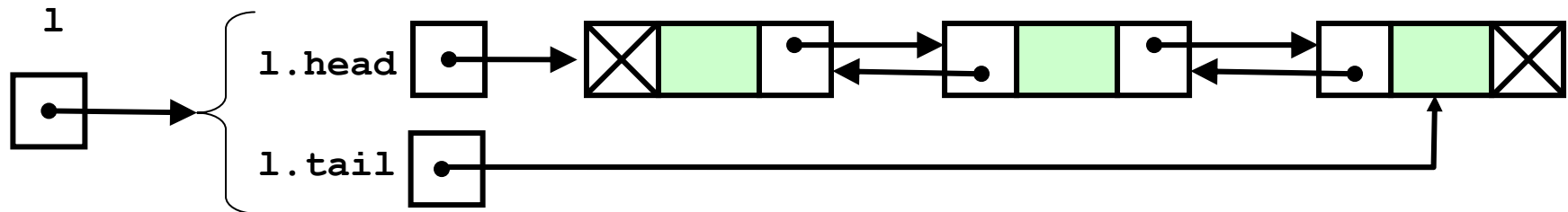
9. Scrivi lo pseudocodice della procedura $\text{COMUNI}(l_1, l_2)$ che ritorna il numero di elementi della lista l_1 che sono anche contenuti nella lista l_2
 - discuti la complessità dell'algoritmo proposto
10. Scrivi lo pseudocodice della procedura non ricorsiva $\text{INVERSA}(l)$ che ritorna una nuova lista singolarmente concatenata in cui gli elementi sono in ordine inverso
11. Scrivi lo pseudocodice della procedura $\text{ACCODA}(l_1, l_2)$ che accoda gli elementi della lista l_2 alla lista l_1 mantenendo l'ordine relativo che gli elementi avevano nelle liste originarie
 - puoi supporre di poter modificare le liste in input

Lista doppiamente concatenata

- Oltre all'attributo `next` i nodi dispongono anche dell'attributo `prev`

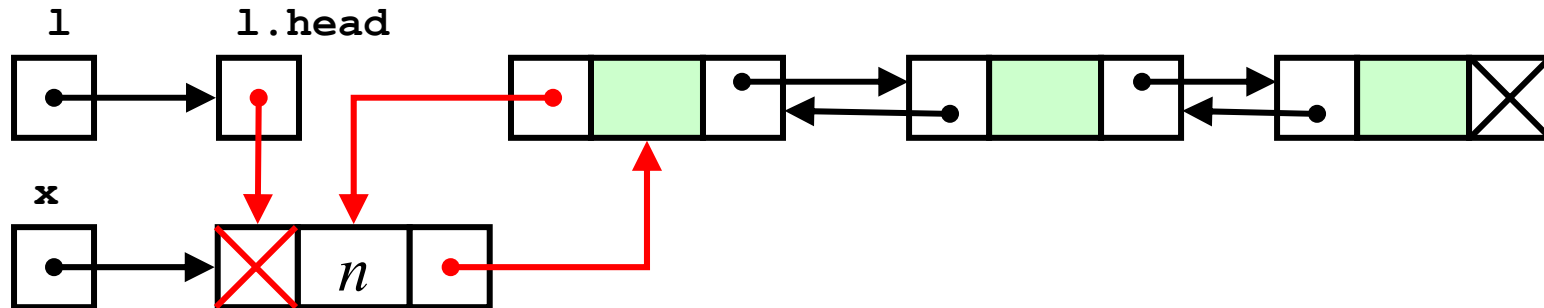
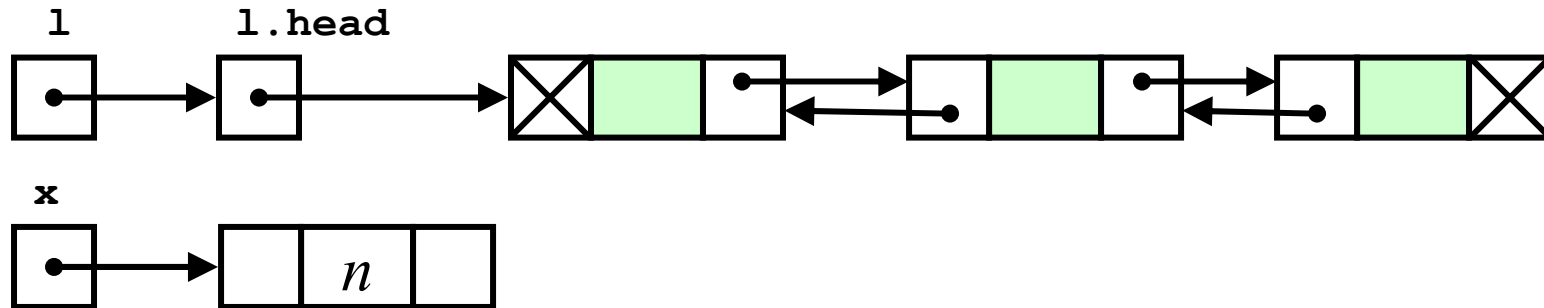


- Talvolta la lista `l` dispone anche di un attributo `l.tail`



Inserimento nella lista

- $\text{INSERT}(l, n)$: inserimento in testa alla lista



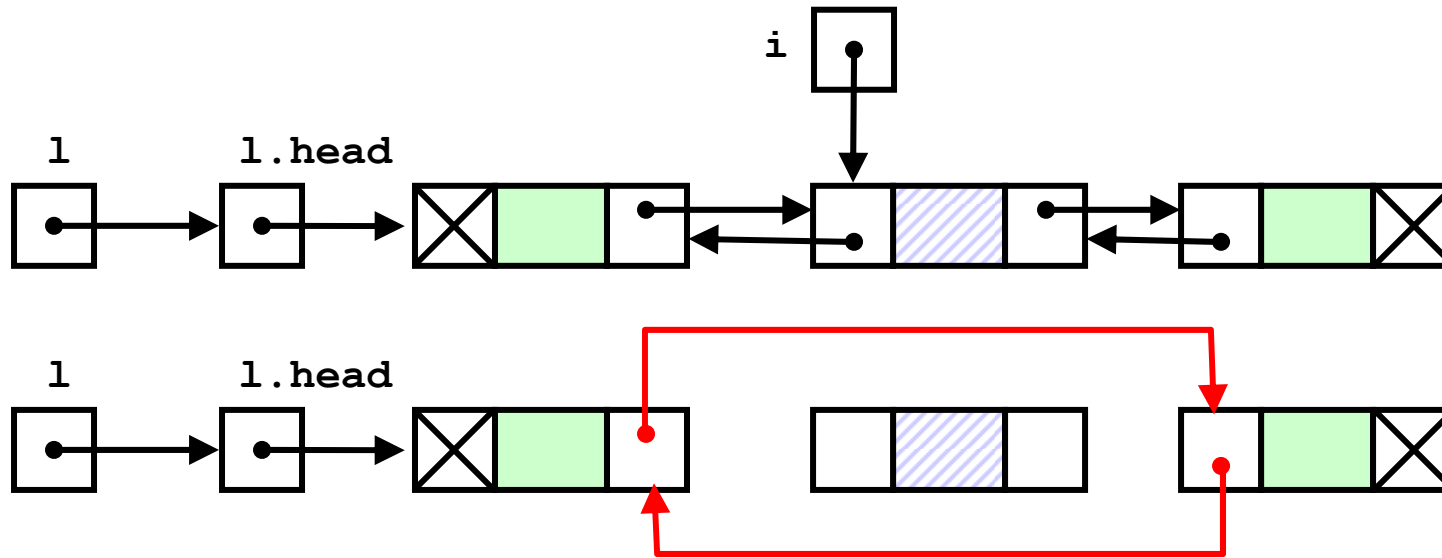
Inserimento nella lista

- `INSERT(l,n)`: inserimento in testa alla lista

INSERT (l,n)

```
1. ▷ x è un nuovo oggetto con tre campi:
2. ▷      x.info (intero)
3. ▷      x.prev, x.next (riferimenti ad oggetti analoghi)
4. x.info = n
5. x.next = l.head
6. x.prev = NULL
7. if l.head != NULL
8.     l.head.prev = x
9. l.head = x
```

Cancellazione di un elemento



```
DELETE(l,i)
```

```
1. if i.prev != NULL
```

```
2.     i.prev.next = i.next
```

```
3. else
```

```
4.     l.head = i.next
```

```
5. if i.next != NULL
```

```
6.     i.next.prev = i.prev
```

Esercizi su liste doppiamente concatenate

12. Scrivi lo pseudocodice dell'operazione

`INSERT_BEFORE(l, n, i)` che riceva come parametri una lista doppiamente concatenata `l`, un intero `n` ed un iteratore `i`, e inserisca `n` nella lista prima dell'elemento riferito da `i`

- discuti la complessità della procedura

13. Scrivi lo pseudocodice dell'operazione

`ADD_AFTER(l, n, i)` che riceva come parametri una lista doppiamente concatenata `l`, un intero `n` ed un iteratore `i`, e inserisca `n` nella lista dopo l'elemento riferito da `i`

- discuti la complessità della procedura

Esercizi su liste doppiamente concatenate

14. Implementa una coda utilizzando una lista doppiamente concatenata

- è possibile che le operazioni ENQUEUE e DEQUEUE abbiano entrambe complessità $\Theta(1)$?
- come si potrebbe fare per ottenere questo risultato?

15. Scrivi lo pseudocodice della procedura DELETE(l, u) che rimuova l'elemento che ha valore u da una lista doppiamente concatenata di interi

- discuti la complessità dell'algoritmo

Esercizi su liste doppiamente concatenate

16. Scrivi lo pseudocodice della procedura `INSERT_ORDERED(l,u)` che inserisca nella lista l (che si suppone ordinata in senso crescente) un intero u mantenendo l'ordinamento crescente della lista
17. Scrivi lo pseudocodice della procedura `MERGE(l_1 , l_2)` che accetti come parametri due liste doppiamente concatenate di interi ordinate in senso crescente e restituisca una lista ordinata in senso crescente con gli elementi di entrambe
 - puoi supporre che tutti gli elementi delle liste siano diversi

Esercizi sulle liste ordinate

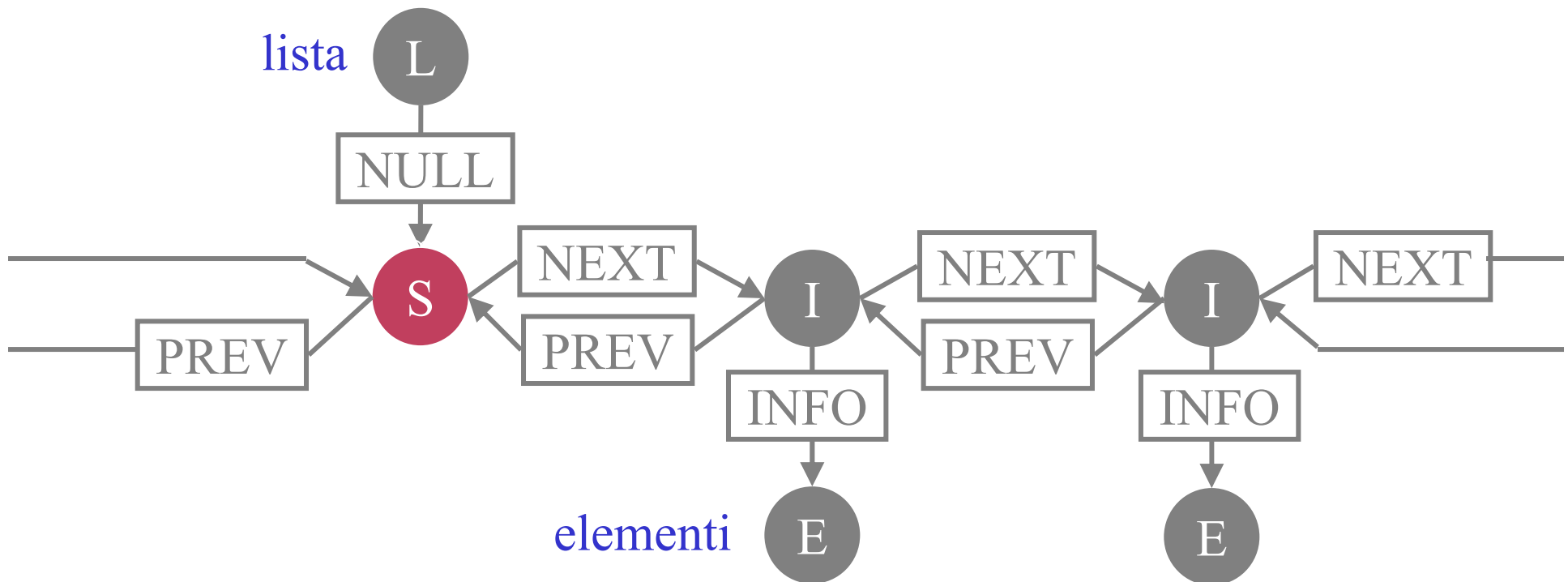
18. Scrivi lo pseudocodice della procedura `DOPPIONI(l)` che verifichi che una lista (non ordinata) doppiamente concatenata di interi non abbia doppioni
19. Scrivi lo pseudocodice della procedura `DOPPIONI_SORTED(l)` che verifichi che una lista doppiamente concatenata di interi ordinata in senso non-decrescente non abbia doppioni

Liste con sentinelle

- Le liste realizzate con oggetti e puntatori offrono l'opportunità di introdurre speciali iteratori chiamati “sentinelle”
- Il primo iteratore della lista (la “sentinella”) è sempre presente e non ha nessun elemento associato
- L'iteratore non valido coincide con l'iteratore che identifica la sentinella
- La struttura dati è circolare

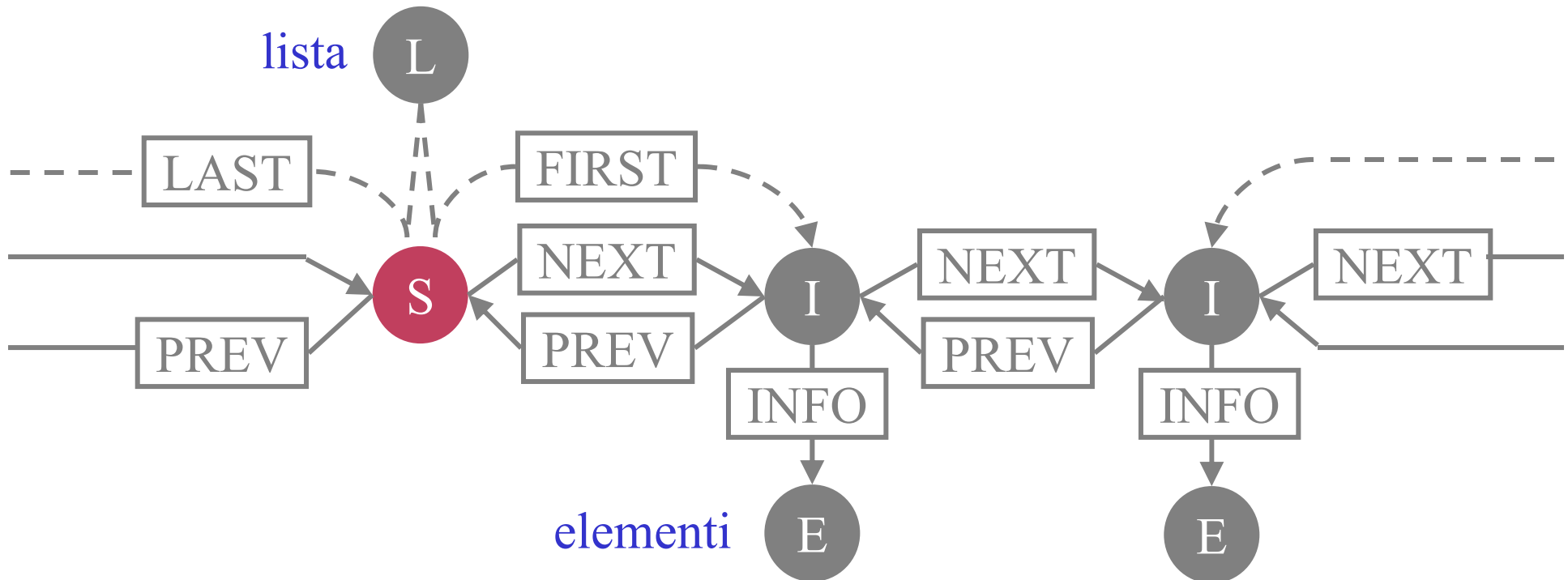
Liste con sentinelle

- Dalla lista si accede direttamente (ed esclusivamente) all'iteratore non-valido, cioè alla sentinella



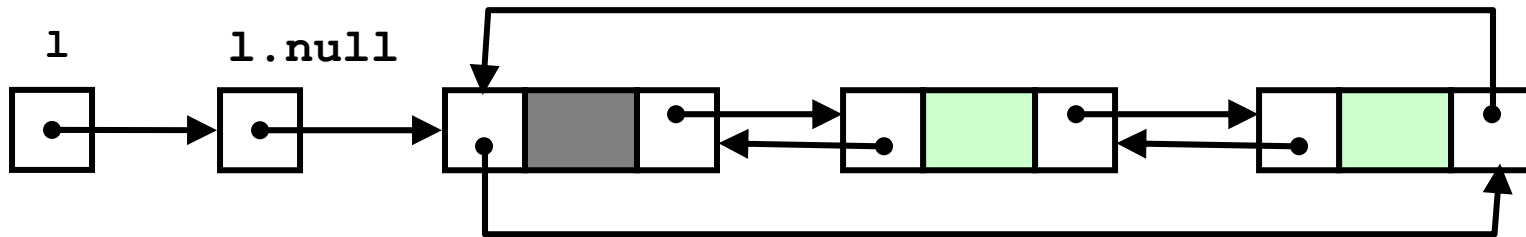
Uso della sentinella

- Concatenando NULL+NEXT si ottiene FIRST
- Concatenando NULL+PREV si ottiene LAST
- Questa strategia comporta diversi altri vantaggi
 - molte procedure risulteranno semplificate

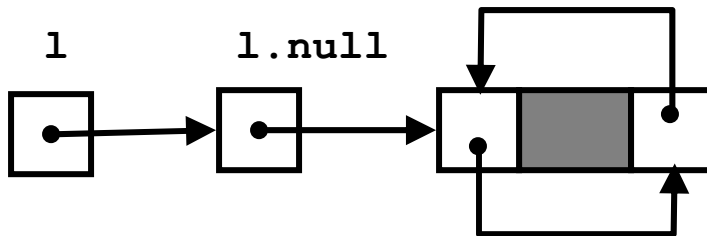


Realizzazione della sentinella

- La sentinella è un nodo fittizio introdotto in testa alla lista



- La lista vuota contiene solamente la sentinella



Procedure nelle liste con sentinelle

- Esempi di procedura semplificata dall'uso di sentinelle
 - lista doppiamente concatenata (senza sentinella)

DELETE(l,i) ▷ versione senza sentinella ($i \neq \text{NULL}$)

```
1. if i.prev != NULL
2.     i.prev.next = i.next
3. else
4.     l.head = i.next
5. if i.next != NULL
6.     i.next.prev = i.prev
```

- lista doppiamente concatenata con sentinella

DELETE(l,i) ▷ versione con sentinella ($i \neq l.\text{null}$)

```
1. i.prev.next = i.next
2. i.next.prev = i.prev
```

Esercizi sulle liste con sentinelle

20. Scrivi lo pseudocodice della procedura `INSERT(l,n)` che inserisce in testa ad una lista con sentinella `l` un intero `n`
21. Scrivi lo pseudocodice della procedura `SEARCH(l,n)` che ritorna un iteratore all'elemento della lista con sentinella `l` che ha valore `n`
 - `SEARCH(l,n)` ritorna `l.null` se `n` non è presente nella lista `l`