Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Corso di Algoritmi e Strutture Dati

Strutture dati elementari

Insiemi dinamici

- Un insieme di elementi che "cambia" nel corso del tempo è detto dinamico
- Un insieme dinamico che supporta le operazioni di
  - Inserimento di un elemento
  - Eliminazione di un elemento
  - Verifica dell'appartenenza di un elemento all'insieme
- si dice dizionario
- Nelle implementazioni tipiche, gli elementi sono oggetti contenenti diversi campi
  - Una chiave e dati satellite
- In alcuni casi, si presuppone l'esistenza di una relazione d'ordine totale tra le chiavi degli elementi

Operazioni tipiche

- Query
- Search (S,k)

  restituisce un "puntatore" x all'elemento con chiave k o il puntatore nullo (NIL) se un tale elemento non esiste
- Minimum (S)

  restituisce un puntatore all'elemento con la chiave più piccola
- Maximum (S)

  restituisce un puntatore all'elemento con la chiave più grande
- Successor (S,x)

  restituisce un puntatore all'elemento la cui chiave è immediatamente maggiore di quella di x
- Predecessor (S,x)

  restituisce un puntatore all'elemento la cui chiave è immediatamente minore di quella di x

Operazioni tipiche

- Operazioni che modificano l'insieme
- Insert (S,x)

  Inserisce in S l'elemento puntato da x
- Delete (S,x)

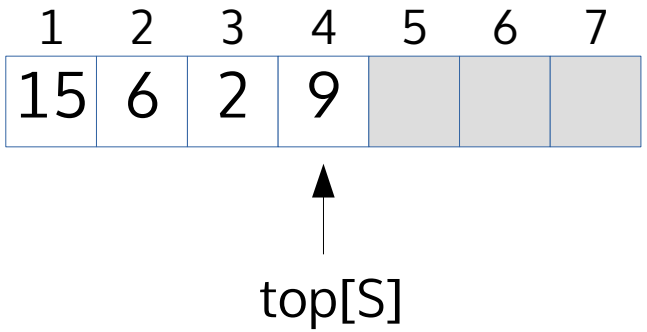
  Rimuove da S l'elemento puntato da x (non si fornisce la chiave)

Stack e Code

- Stack e code sono strutture dati in cui l'operazione DELETE rimuove un elemento in maniera predefinita
- In uno stack, l'elemento rimosso è quello più recentemente inserito
  - Strategia LIFO (Last In First Out)
- In una coda, l'elemento rimosso è quello inserito da più tempo
  - Strategia FIFO (First In First Out)
- Vediamo come usare un semplice array per implementare stack e code

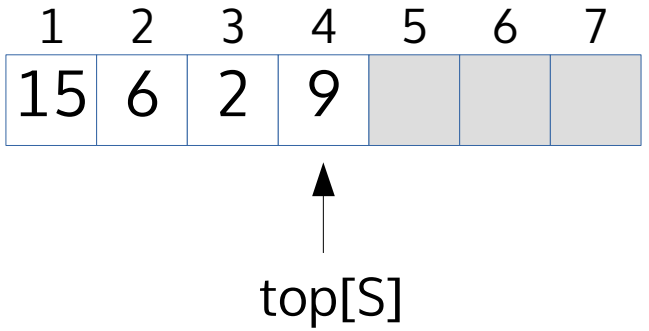
Stack

- L'operazione Insert è tipicamente chiamata Push
- L'operazione Delete è tipicamente chiamata Pop
- Possiamo implementare uno stack di al più n elementi mediante un array
  - Di lunghezza n
  - Con un attributo top che indica la posizione dell'elemento inserito più di recente (0 se vuoto)



Stack

```
Stack-Empty (S)
if top[S]=0
    then return TRUE
    else return FALSE
```



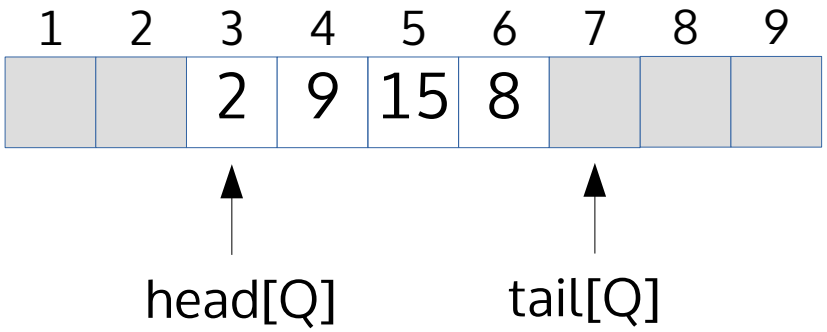

```
Push (S,x)
top[S] ← top[S] + 1
S[top[S]] ← x
```

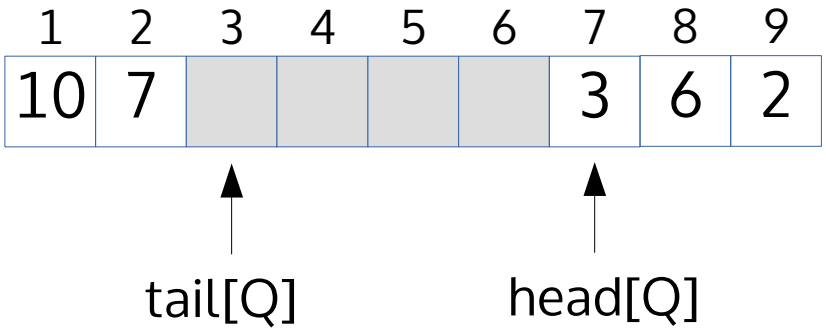
```
Pop (S)
if Stack-Empty(S)
    then error “underflow”
    else top[S] ← top[S]-1
        return S[top[S]+1]
```

- Queste operazioni richiedono O(1)
- Trascuriamo la verifica dell'overflow

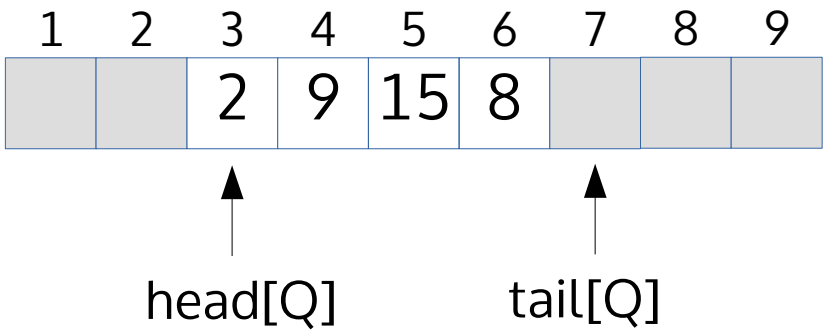
Code

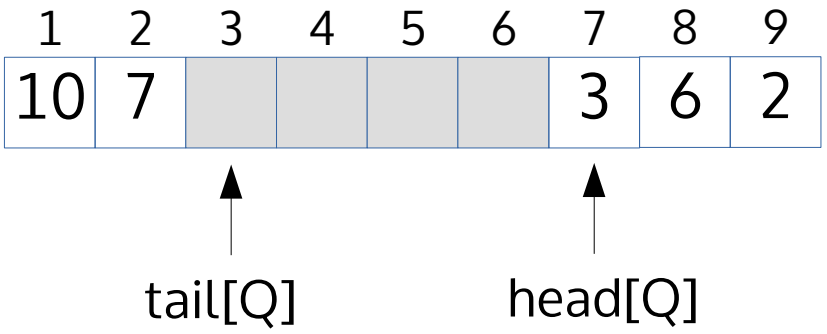
- L'operazione Insert è tipicamente chiamata Enqueue
- L'operazione Delete è tipicamente chiamata Dequeue
- Possiamo implementare una coda di n-1 elementi mediante un array di lunghezza n
  - Con un attributo head che indica la posizione della testa
  - Con un attributo tail che indica la posizione in cui inserire il prossimo elemento accodato
  - Coda vuota se head=tail, piena se head=tail+1





Code





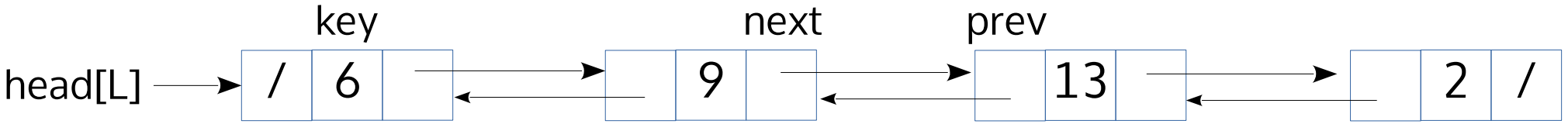

```
Enqueue (Q,x)
Q[tail[Q]] ← x
if tail[Q] = length[Q]
    then tail[Q] ← 1
    else tail[Q] ← tail[Q]+1
```

```
Dequeue (Q)
x ← Q[head[Q]]
if head[Q] = length[Q]
    then head[Q] ← 1
    else head[Q] ← head[Q]+1
return x
```

- Queste operazioni richiedono O(1)
- Trascuriamo la verifica dell'overflow e dell'underflow

Liste concatenate

- Una lista concatenata è una struttura dati in cui gli elementi sono disposti secondo un ordine lineare
- A differenza degli array, tale ordine è determinato da puntatori
- Supportano tutte le operazioni indicate per gli insiemi dinamici
  - Anche se non necessariamente in maniera efficiente
- Una lista può essere singolarmente concatenata, ordinata, circolare
- Consideriamo liste doppiamente concatenate e non ordinate



Liste concatenate: ricerca

- Ricerca un elemento con chiave k e restituisce un puntatore a tale elemento

```
List-Search (L,k)
x ← head[L]
while x≠NIL and key[x]≠k
    do x ← next[x]
return x
```

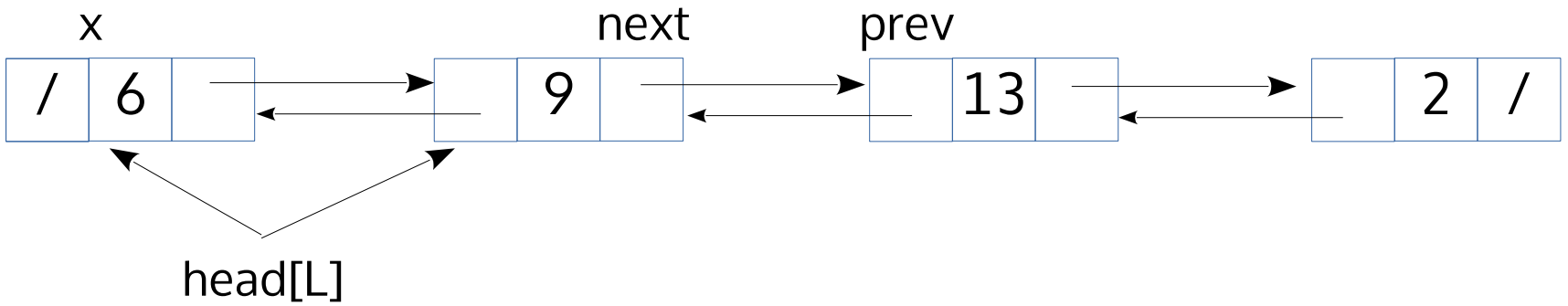
- Restituisce NIL se non c'è un elemento con chiave k
- Richiede un tempo $\Theta(n)$ nel caso peggiore

Liste concatenate: inserimento

- Inserimento di un elemento in testa alla lista

```
List-Insert (L,x)
next[x] ← head[L]
if head[L]≠NIL
    then prev[head[L]] ← x
head[L] ← x
prev[x] ← NIL
```

- Richiede un tempo costante O(1)

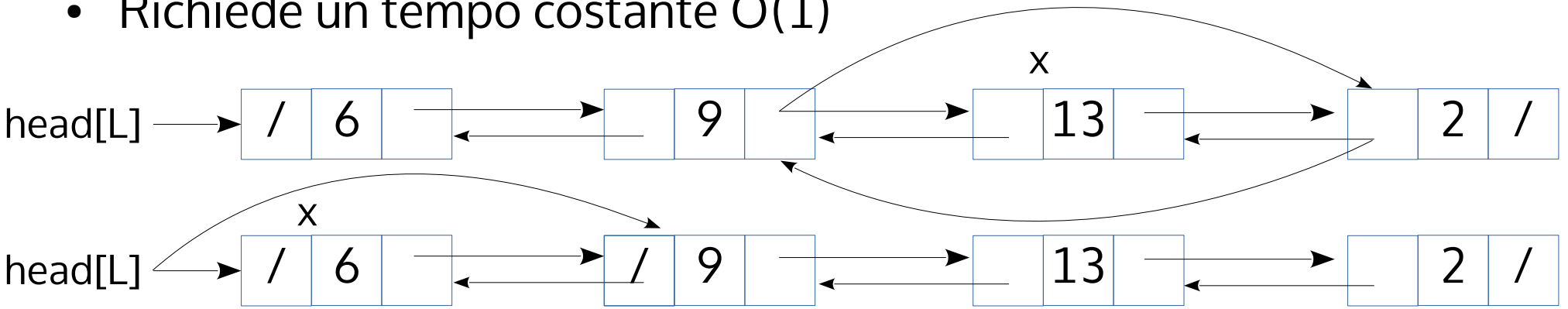


Liste concatenate: eliminazione

- Elimina l'elemento individuato dal puntatore fornito
  - Eventualmente restituito dalla funzione List-Search

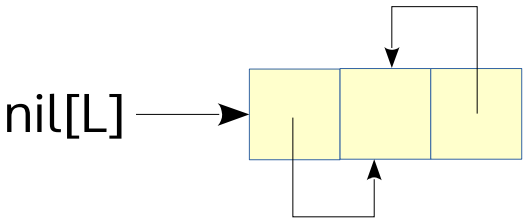
```
List-Delete (L,x)
if prev[x]≠NIL
    then next[prev[x]] ← next[x]
    else head[L] ← next[x]
if next[x]≠NIL
    then prev[next[x]] ← prev[x]
```

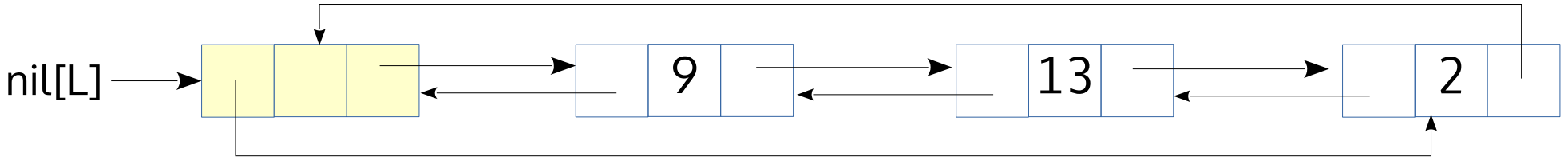
- Richiede un tempo costante O(1)



Liste concatenate con sentinella

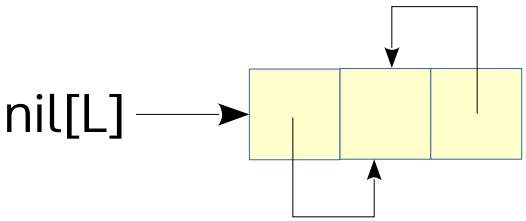
- Il codice delle funzioni viste può essere semplificato assumendo la presenza di un oggetto *sentinella* come oggetto "vuoto"
- La lista diventa circolare
- Aggiungiamo alla lista un attributo nil che punta alla sentinella
- Una lista vuota è costituita dalla sola sentinella

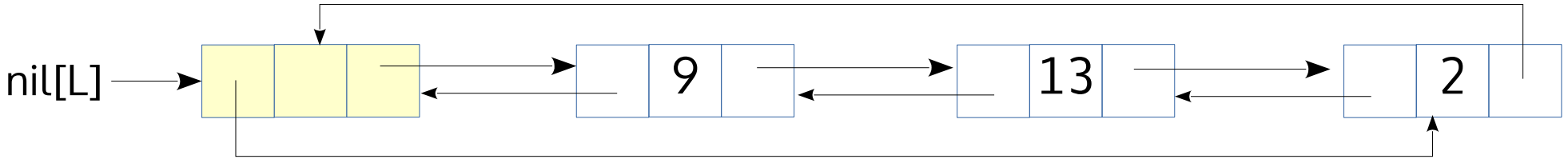




Liste concatenate con sentinella

- Nel codice
  - NIL può essere sostituito da nil[L]
  - head[L] può essere sostituito da next[nil[L]]

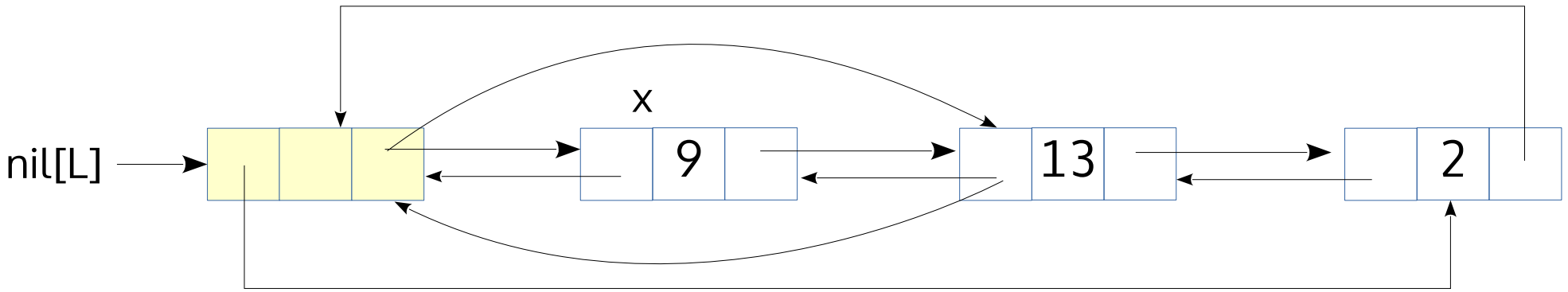




Liste concatenate: eliminazione

```
List-Delete (L,x)
if prev[x]≠NIL
    then next[prev[x]] ← next[x]
    else head[L] ← next[x]
if next[x]≠NIL
    then prev[next[x]] ← prev[x]
```

```
List-Delete' (L,x)
next[prev[x]] ← next[x]
prev[next[x]] ← prev[x]
```



Liste concatenate: ricerca

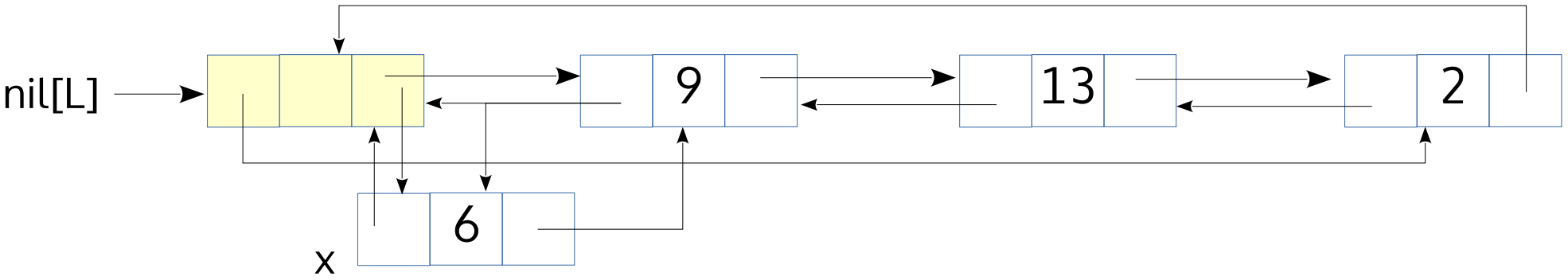
```
List-Search (L,k)
x ← head[L]
while x≠NIL and key[x]≠k
    do x ← next[x]
return x
```

```
List-Search' (L,k)
x ← next[nil[L]]
while x≠nil[L] and key[x]≠k
    do x ← next[x]
return x
```

Liste concatenate: inserimento

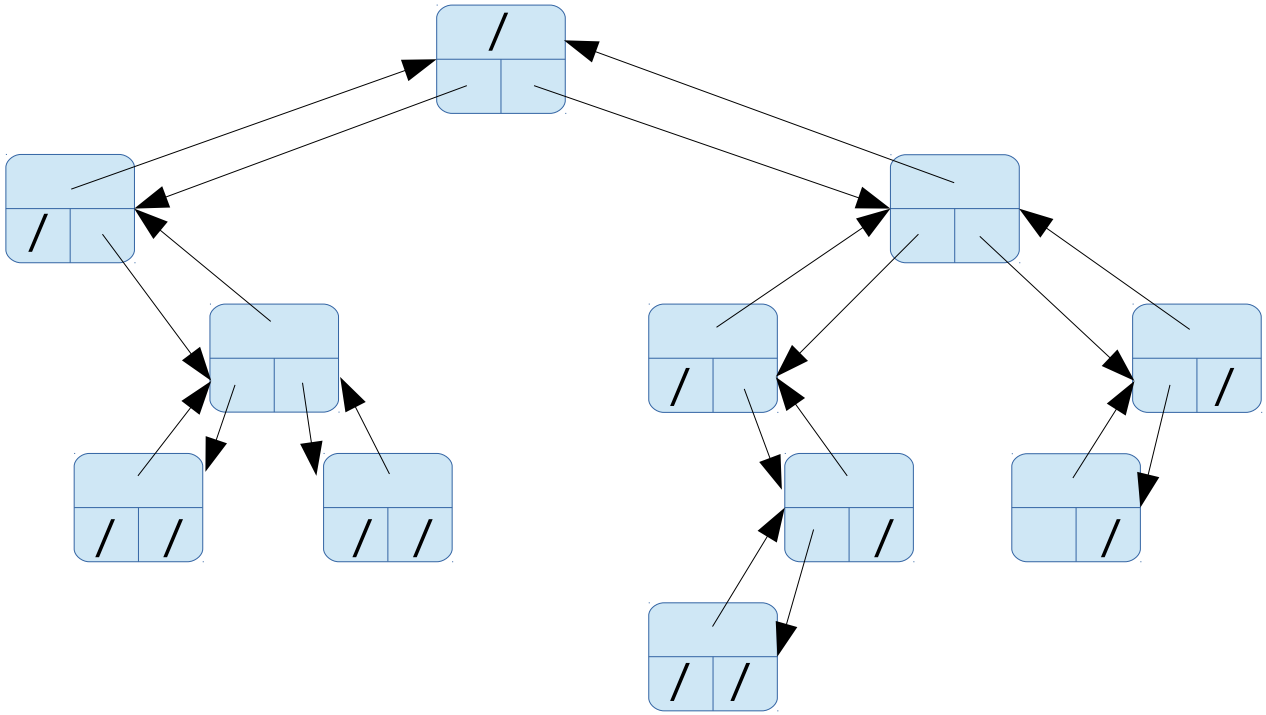
```
List-Insert (L,x)
next[x] ← head[L]
if head[L]≠NIL
    then prev[head[L]] ← x
head[L] ← x
prev[L] ← NIL
```

```
List-Insert' (L,x)
next[x] ← next[nil[L]]
prev[next[nil[[L]]] ← x
next[nil[[L]] ← x
prev[x] ← nil[L]
```



Rappresentazioni di alberi binari

- Un albero binario può essere rappresentato mediante una struttura dati concatenata
  - Ogni nodo ha un puntatore al padre (p), al figlio di sinistra (left) e al figlio di destra (right)
  - root[T] è l'attributo che fornisce un puntatore alla radice dell'albero



Rappresentazioni di alberi n-ari

- Un albero con un numero non limitato di figli per nodo può essere rappresentato mediante una struttura dati concatenata
  - Ogni nodo ha un puntatore al padre (p), al figlio di sinistra (left-child) e al fratello di destra (right-sibling)
  - root[T] è l'attributo che fornisce un puntatore alla radice dell'albero