

# Strutture dati elementari

Vittorio Maniezzo - Università di Bologna

1

## Pile (Stacks)

---

**Dati:** un insieme  $S$  di elementi.

**Operazioni:** PUSH, POP

- **PUSH:** inserisce un elemento in  $S$
- **POP:** restituisce l'ultimo elemento inserito e lo rimuove da  $S$

**Politica:** Last-In-First-Out (LIFO)

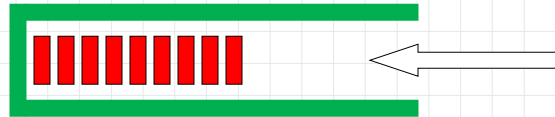
Vittorio Maniezzo - Università di Bologna

2

2

# Pila

PUSH...



```
PUSH(S, o)
  S.top = S.top + 1
  S[S.top] = o
```

3

# Pila

POP...



```
POP(S)
  o = S[S.top]
  S.top = S.top - 1
  return o
```

4

## Code (Queues)

**Dati:** un insieme  $S$  di elementi.

**Operazioni:** ENQUEUE, DEQUEUE

- **ENQUEUE** : inserisce un elemento in  $S$
- **DEQUEUE** : restituisce l'elemento da più tempo presente (il più vecchio) e lo rimuove da  $S$

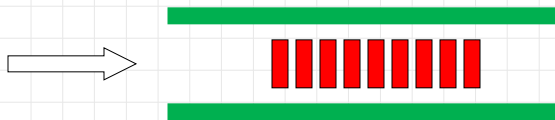
**Politica:** First-In-First-Out (FIFO)

Implementazione con **vettori circolari**:

- $Q.head$  indica la posizione della testa (la prima *usata*)
- $Q.tail$  indica la prima posizione a destra della coda (la prima *libera*)
- Se la coda è vuota  $Q.head = Q.tail$
- Inizialmente,  $Q.head = Q.tail = 1$

## Coda

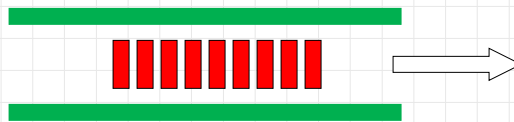
**ENQUEUE . . .**



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

## Coda

DEQUEUE . . .



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

## Problemi con i vettori

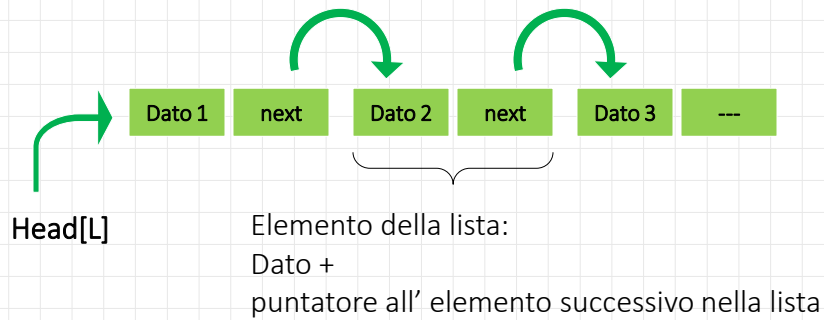
Vettori

- Semplici,
- Veloci

ma

- Bisogna specificare la lunghezza staticamente
- Legge di Murphy:  
Se usi un vettore di lunghezza  $n$  = doppio di ciò che ti serve,  
domani avrai bisogno di un vettore lungo  $n+1$
- Esiste una struttura dati più flessibile?

## Lista concatenata semplice

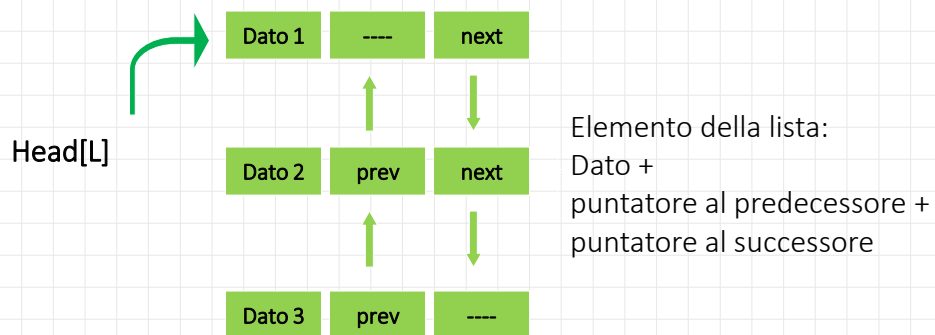


Vittorio Maniezzo - Università di Bologna

9

9

## Lista doppiamente concatenata



Vittorio Maniezzo - Università di Bologna

10

10

## Ricerca e Inserimento

```
LIST-SEARCH(L,k)
  x = head[L]
  while x != nil and key[x] != k
    do x = next[x]
  return x
```

```
LIST-INSERT(L,x)
  next[x] = head[L]
  if head[L] != nil
  then prev[head[L]] = x
  head[L] = x
  prev[x] = nil
```

Vittorio Maniezzo - Università di Bologna

11

11

## Cancellazione

```
LIST-DELETE(L,k)

  x = LIST-SEARCH(L,k)
  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]
```

Vittorio Maniezzo - Università di Bologna

12

12

## Costi Computazionali

|               |             |             |
|---------------|-------------|-------------|
| Inserimento   | LIST-INSERT | $\Theta(1)$ |
| Cancellazione | LIST-DELETE | $\Theta(1)$ |
| Ricerca       | LIST-SEARCH | $\Theta(n)$ |

Se non conto il  
ListSearch interno

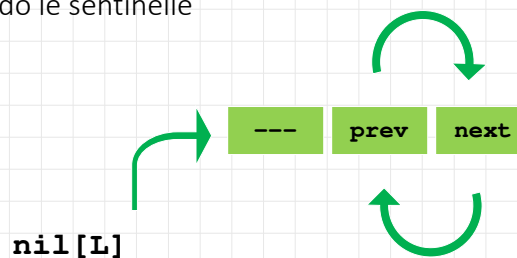
Vittorio Maniezzo - Università di Bologna

13

13

## Sentinelle

Lista vuota usando le sentinelle



Una sentinella è un oggetto artificiale che semplifica la gestione dei limiti dei dati.

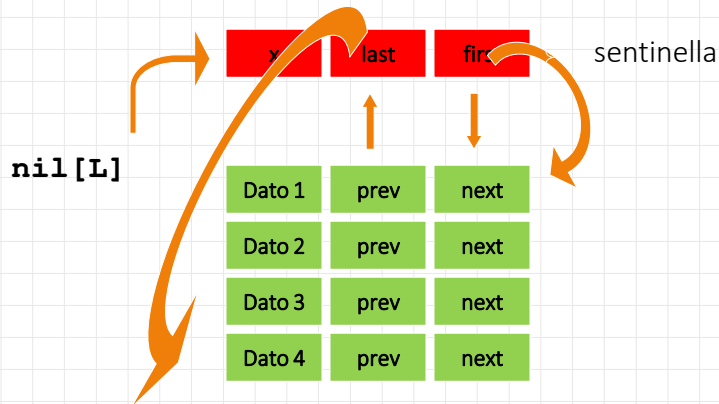
- `nil[L]` contiene `nil`, è l'inizio
- `next[nil[L]]` punta alla testa della lista
- La lista vuota è la sola sentinella, `nil[L]`
- `head[L]` non è più necessario

Vittorio Maniezzo - Università di Bologna

14

14

## Sentinelle



Vittorio Maniezzo - Università di Bologna

15

15

## Cancellazione con le sentinelle

LIST-DELETE(L,k)

```
x = LIST-SEARCH[L,k]
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]
```

Senza  
sentinelle

Con la  
sentinella

LIST-DELETE-SENTINEL(L,k)

```
x = LIST-SEARCH[L,k]
next[prev[x]] = next[x]
prev[next[x]] = prev[x]
```

Esercizio: usare le sentinelle per  
SEARCH e INSERT

Vittorio Maniezzo - Università di Bologna

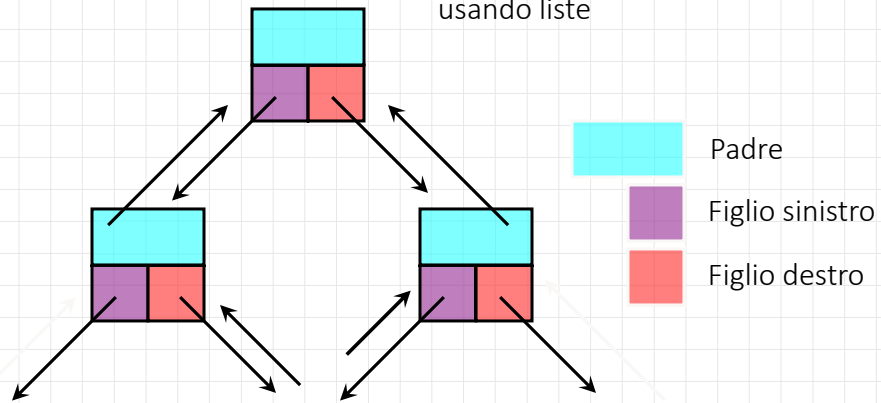
16

16



## Alberi binari e liste

Alberi binari rappresentati  
usando liste

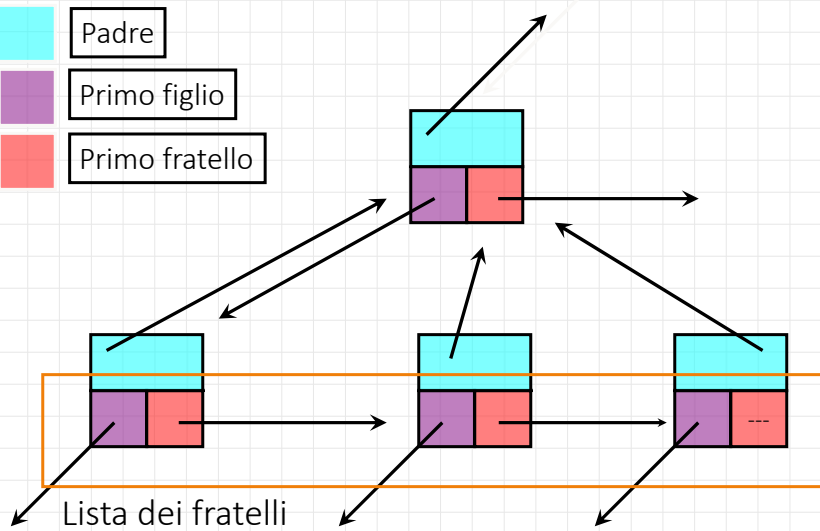


Vittorio Maniezzo - Università di Bologna

17

17

## Alberi generali e liste



Vittorio Maniezzo - Università di Bologna

18

18

## Costo computazionale delle operazioni sulle liste

|               | Singly linked<br>non ordinata | Singly<br>linked<br>ordinata | Doubly<br>linked<br>non<br>ordinata | Doubly<br>linked<br>ordinata |
|---------------|-------------------------------|------------------------------|-------------------------------------|------------------------------|
| Ricerca       |                               |                              |                                     |                              |
| Inserimento   |                               |                              |                                     |                              |
| Cancellazione |                               |                              |                                     |                              |
| Successore    |                               |                              |                                     |                              |
| Predecessore  |                               |                              |                                     |                              |
| Massimo       |                               |                              |                                     |                              |