

# Algoritmi e Strutture Dati

## Capitolo 3

### Strutture dati elementari

Camil Demetrescu, Irene Finocchi,  
Giuseppe F. Italiano

# Gestione di collezioni di oggetti

Tipo di dato:

- Specifica delle operazioni di interesse su una collezione di oggetti (es. inserisci, cancella, cerca)

Struttura dati:

- Organizzazione dei dati che permette di supportare le operazioni di un tipo di dato usando meno risorse di calcolo possibile

# Il tipo di dato Dizionario

**tipo** Dizionario:

**dati:**

un insieme  $S$  di coppie  $(elem, chiave)$ .

**operazioni:**

$insert(elem\ e, chiave\ k)$

aggiunge a  $S$  una nuova coppia  $(e, k)$ .

$delete(chiave\ k)$

cancella da  $S$  la coppia con chiave  $k$ .

$search(chiave\ k) \rightarrow elem$

se la chiave  $k$  è presente in  $S$  restituisce l'elemento  $e$  ad essa associato,  
e null altrimenti.

# Il tipo di dato Pila

**tipo** Pila:

**dati:**

una sequenza  $S$  di  $n$  elementi.

**operazioni:**

$\text{isEmpty}() \rightarrow \text{result}$

restituisce `true` se  $S$  è vuota, e `false` altrimenti.

$\text{push}(\text{elem } e)$

aggiunge  $e$  come ultimo elemento di  $S$ .

$\text{pop}() \rightarrow \text{elem}$

toglie da  $S$  l'ultimo elemento e lo restituisce.

$\text{top}() \rightarrow \text{elem}$

restituisce l'ultimo elemento di  $S$  (senza toglierlo da  $S$ ).

# Il tipo di dato Coda

**tipo** Coda:

**dati:**

una sequenza  $S$  di  $n$  elementi.

**operazioni:**

$\text{isEmpty}() \rightarrow \text{result}$

restituisce `true` se  $S$  è vuota, e `false` altrimenti.

$\text{enqueue}(\text{elem } e)$

aggiunge  $e$  come ultimo elemento di  $S$ .

$\text{dequeue}() \rightarrow \text{elem}$

toglie da  $S$  il primo elemento e lo restituisce.

$\text{first}() \rightarrow \text{elem}$

restituisce il primo elemento di  $S$  (senza toglierlo da  $S$ ).

# Tecniche di rappresentazione dei dati

Rappresentazioni indicizzate:

- I dati sono contenuti in array

Rappresentazioni collegate:

- I dati sono contenuti in record collegati fra loro mediante puntatori

# Pro e contro

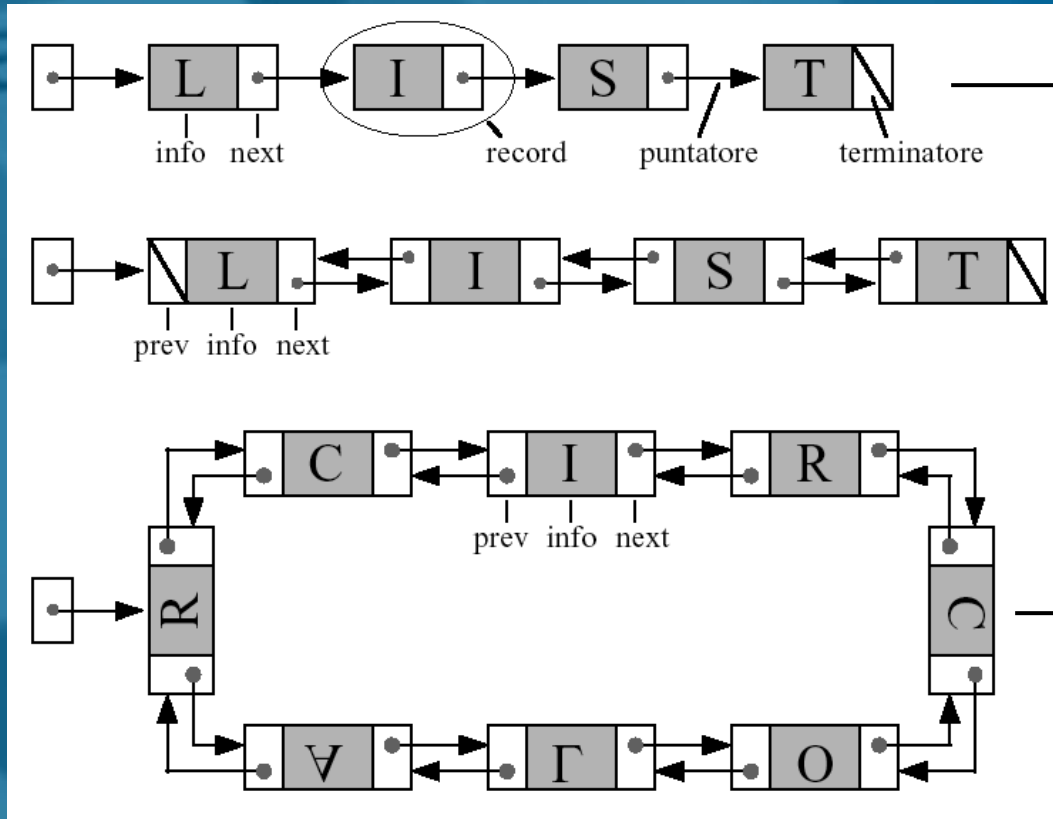
## Rappresentazioni indicizzate:

- **Pro:** accesso diretto ai dati mediante indici
- **Contro:** dimensione fissa (riallocazione array richiede tempo lineare)

## Rappresentazioni collegate:

- **Pro:** dimensione variabile (aggiunta e rimozione record in tempo costante)
- **Contro:** accesso sequenziale ai dati

# Esempi di strutture collegate



Lista semplice

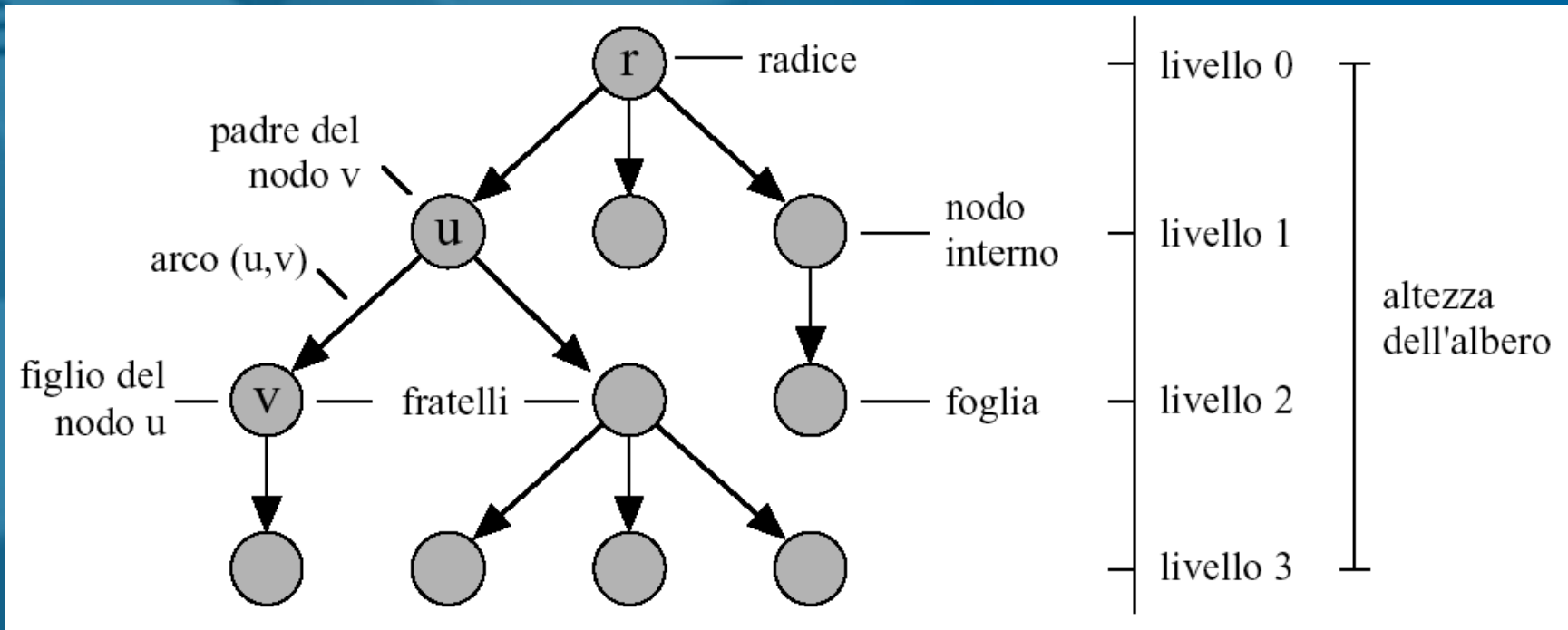
Lista doppiamente collegata

Lista circolare doppiamente collegata



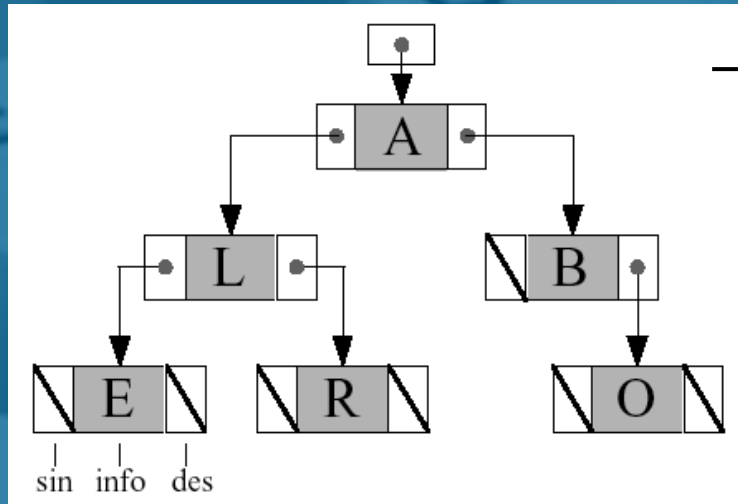
# Alberi

## Organizzazione gerarchica dei dati



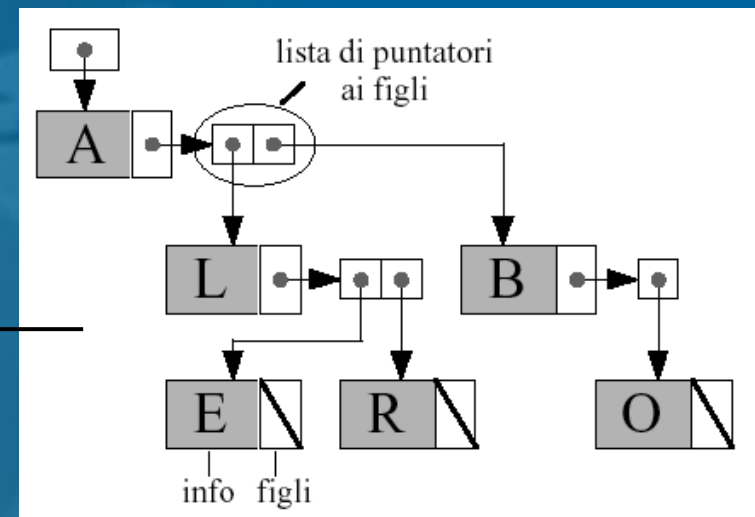
Dati contenuti nei **nodi**, relazioni gerarchiche definite dagli **archi** che li collegano

# Rappresentazioni collegate di alberi

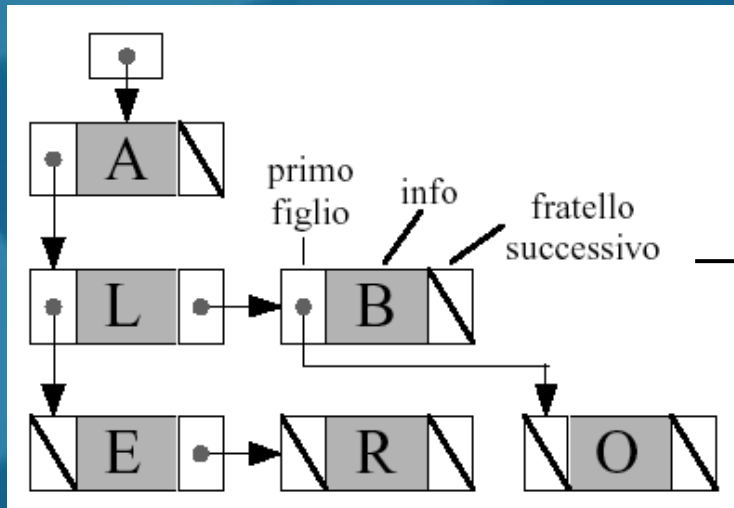


Rappresentazione con puntatori ai figli (nodi con numero limitato di figli)

Rappresentazione con liste di puntatori ai figli (nodi con numero arbitrario di figli)



# Rappresentazioni collegate di alberi



Rappresentazione  
di tipo primo figlio-  
fratello successivo  
(nodi con numero  
arbitrario di figli)

# Visite di alberi

Algoritmi che consentono l'**accesso sistematico ai nodi e agli archi** di un albero

Gli algoritmi di visita si distinguono in base al particolare ordine di accesso ai nodi

# Algoritmo di visita generica

*visitaGenerica* visita il nodo **r** e tutti i suoi discendenti in un albero

```

algoritmo visitaGenerica(nodo  $r$ )
1.    $S \leftarrow \{r\}$ 
2.   while ( $S \neq \emptyset$ ) do
3.       estrai un nodo  $u$  da  $S$ 
4.       visita il nodo  $u$ 
5.        $S \leftarrow S \cup \{ \text{figli di } u \}$ 
    
```

Richiede tempo  **$O(n)$**  per visitare un albero con  **$n$**  nodi a partire dalla radice

# Algoritmo di visita in profondità

L'algoritmo di visita in profondità (DFS) parte da  $r$  e procede visitando nodi di figlio in figlio fino a raggiungere una foglia. Retrocede poi al primo antenato che ha ancora figli non visitati (se esiste) e ripete il procedimento a partire da uno di quei figli.

# Algoritmo di visita in profondità

## Versione iterativa (per alberi binari):

```
algoritmo visitaDFS(nodo  $r$ )  
  Pila  $S$   
   $S.push(r)$   
  while (not  $S.isEmpty()$ ) do  
     $u \leftarrow S.pop()$   
    if ( $u \neq null$ ) then  
      visita il nodo  $u$   
       $S.push(\text{figlio destro di } u)$   
       $S.push(\text{figlio sinistro di } u)$ 
```

# Algoritmo di visita in profondità

## Versione ricorsiva (per alberi binari):

```
algoritmo visitaDFSRicorsiva(nodo  $r$ )  
1.   if ( $r = \text{null}$ ) then return  
2.   visita il nodo  $r$   
3.   visitaDFSRicorsiva(figlio sinistro di  $r$ )  
4.   visitaDFSRicorsiva(figlio destro di  $r$ )
```



# Algoritmo di visita in ampiezza

L'algoritmo di visita in ampiezza (BFS) parte da  $r$  e procede visitando nodi per livelli successivi. Un nodo sul livello  $i$  può essere visitato solo se tutti i nodi sul livello  $i-1$  sono stati visitati.

# Algoritmo di visita in ampiezza

Versione iterativa (per alberi binari):

```
algoritmo visitaBFS(nodo  $r$ )  
  Coda  $C$   
   $C.enqueue(r)$   
  while (not  $C.isEmpty()$ ) do  
     $u \leftarrow C.dequeue()$   
    if ( $u \neq null$ ) then  
      visita il nodo  $u$   
       $C.enqueue(\text{figlio sinistro di } u)$   
       $C.enqueue(\text{figlio destro di } u)$ 
```

# Riepilogo

- Nozione di **tipo di dato** come specifica delle operazioni su una collezione di oggetti
- **Rappresentazioni indicizzate e collegate** di collezioni di dati: pro e contro
- **Organizzazione gerarchica** dei dati mediante alberi
- **Rappresentazioni collegate classiche di alberi**
- **Algoritmi di esplorazione sistematica dei nodi di un albero** (algoritmi di visita)