Modulo: Strutture dati dinamiche gerarchiche [P2_09]

Unità didattica: strutture dati per la rappresentazione di un albero binario [3-T]

Titolo: Struttura dei dati ed algoritmi di visita

Argomenti trattati:

- ✓ Regole di rappresentazione di un albero binario mediante array
- ✓ Rappresentazione di un albero binario mediante lista multipla
- ✓ Algoritmi di visita iterativi (con stack esplicito) di un albero binario
- ✓ Algoritmi di visita ricorsivi (con stack implicito) di un albero binario

Prerequisiti richiesti: generalità sugli alberi binari

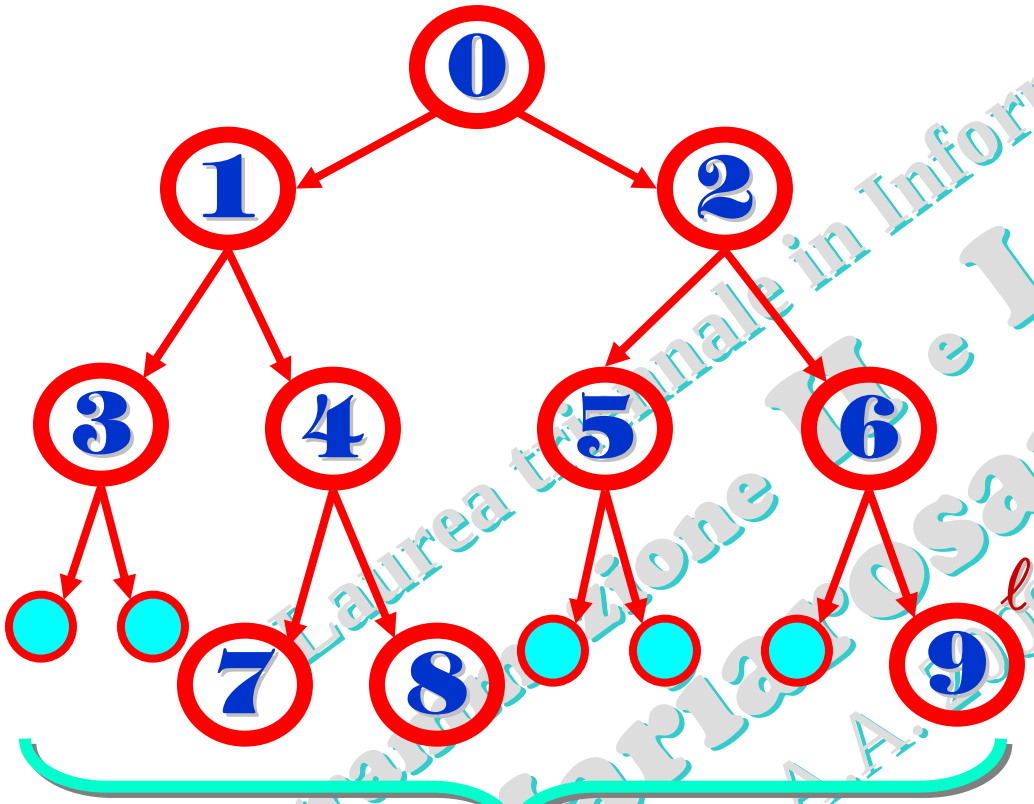
Tipo strutturato albero binario ... in C

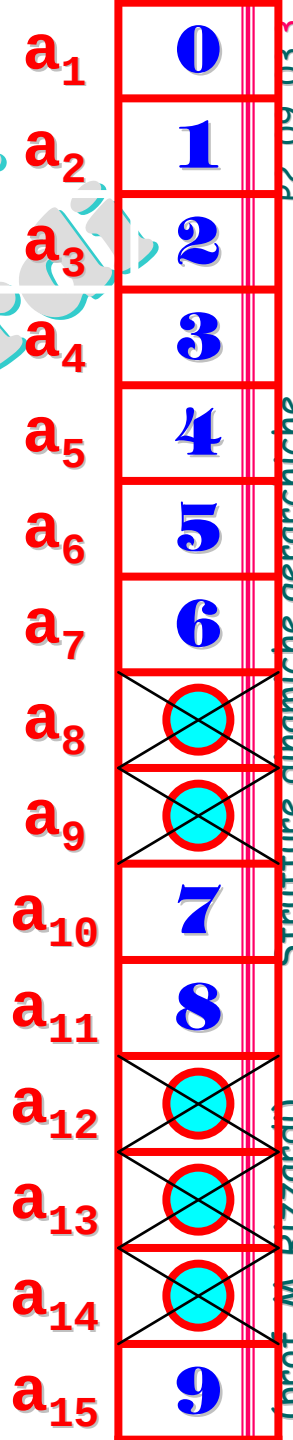
rappresentazione in memoria

albero binario (binary tree) ⇐

- `array`
- **lista multipla** (`struct`)

Esempio:
albero binario mediante array





ℓ = livelli dell'albero

L'albero dev'essere completo

cardinalità dell'array

$= 2^{\ell} - 1$

Regole di rappresentazione di un albero binario tramite array

Un ***albero binario completo*** con $n=2^p-1$ nodi è rappresentato tramite un array $\underline{a} \equiv (a_1, a_2, \ldots, a_n)$ e per un qualsiasi nodo a_i si ha:

- Se $i \neq 1$ ⇒ *padre*$(a_i) \equiv a_k$ dove $k = \lfloor i/2 \rfloor$;
  se $i=1$ allora a_i è radice e non ha padre.
- Se $2i \leq n$ ⇒ *figlio_sinistro*$(a_i) \equiv a_k$ dove $k=2i$;
  se $2i>n$ allora a_i non ha figlio sinistro.
- Se $2i+1 \leq n$ ⇒ *figlio_destro*$(a_i) \equiv a_k$ dove $k=2i+1$;
  se $2i+1>n$ allora a_i non ha figlio destro.

Attenzione!!! In *C* gli indici su array partono da 0, ma se si usa il valore i=0, non si può trovare il figlio sinistro perché 2i=0 (... e nemmeno il destro)!

SOLUZIONE 1: Traslare di ±1 le formule degli indici: ad esempio

- Se $i \neq 1$ ⇨ $padre(a_i) \equiv a_k$ dove $k = \lfloor i/2 \rfloor$;
  se $i=1$ allora a_i è radice e non ha padre.

diventa in *C*

- Se $i+1 \neq 1$ ⇨ $padre(a_{[i]}) \equiv a_{[k-1]}$ dove $k = \lfloor (i+1)/2 \rfloor$;
  se $i+1=1$ allora $a_{[i]}$ è radice e non ha padre.

SOLUZIONE 2: Sovradimensionare di 1 l'array e <u>non usare</u> la prima componente

~~a_0~~	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}

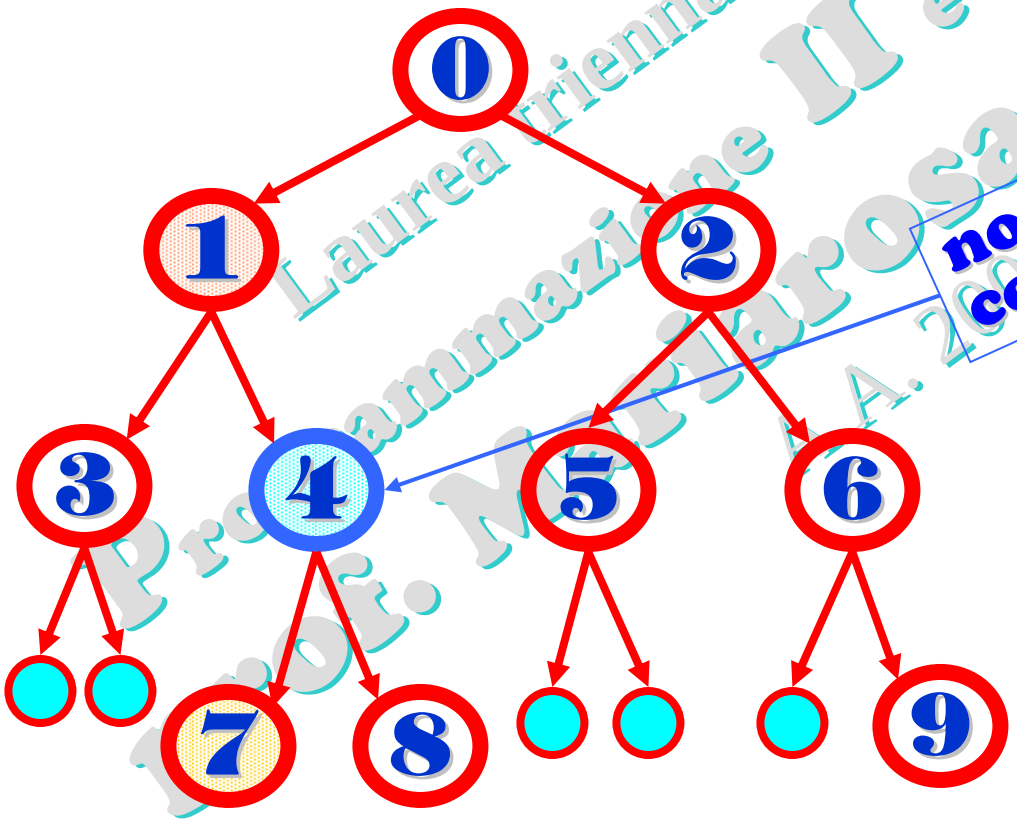
Esempio

nodo corrente a_5

padre di $[a_5] \equiv a_k$ $\quad k = \lfloor 5/2 \rfloor = 2$

figlio sinistro di $[a_5] \equiv a_k$ $\quad k = 2 \cdot 5 = 10$

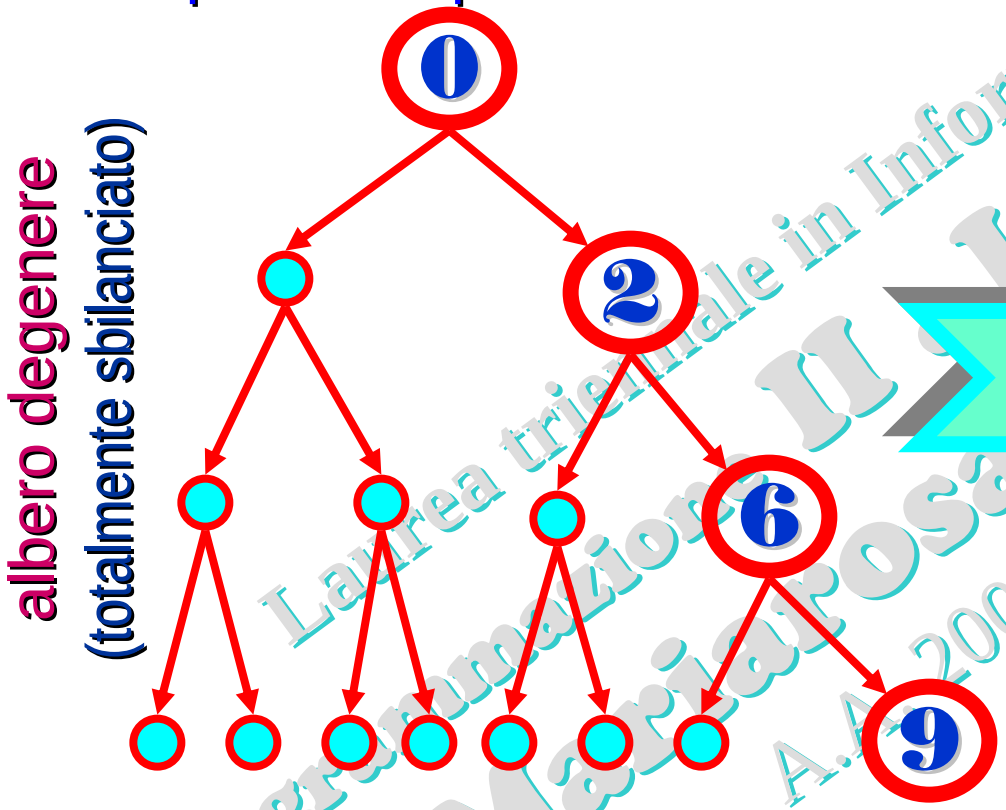
figlio destro di $[a_5] \equiv a_k$ $\quad k = 2 \cdot 5 + 1 = 11$



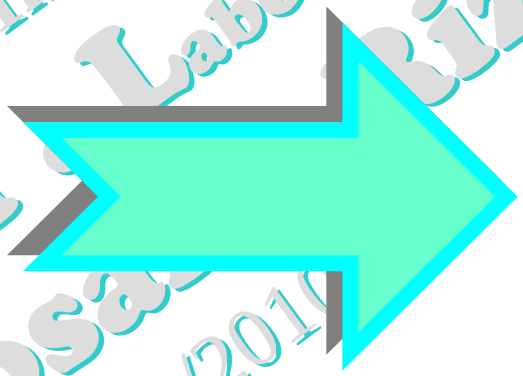
nodo corrente = 4 = a_5

a_1	0
a_2	1
a_3	2
a_4	3
a_5	4
a_6	5
a_7	6
a_8	
a_9	
a_{10}	7
a_{11}	8
a_{12}	
a_{13}	
a_{14}	
a_{15}	9

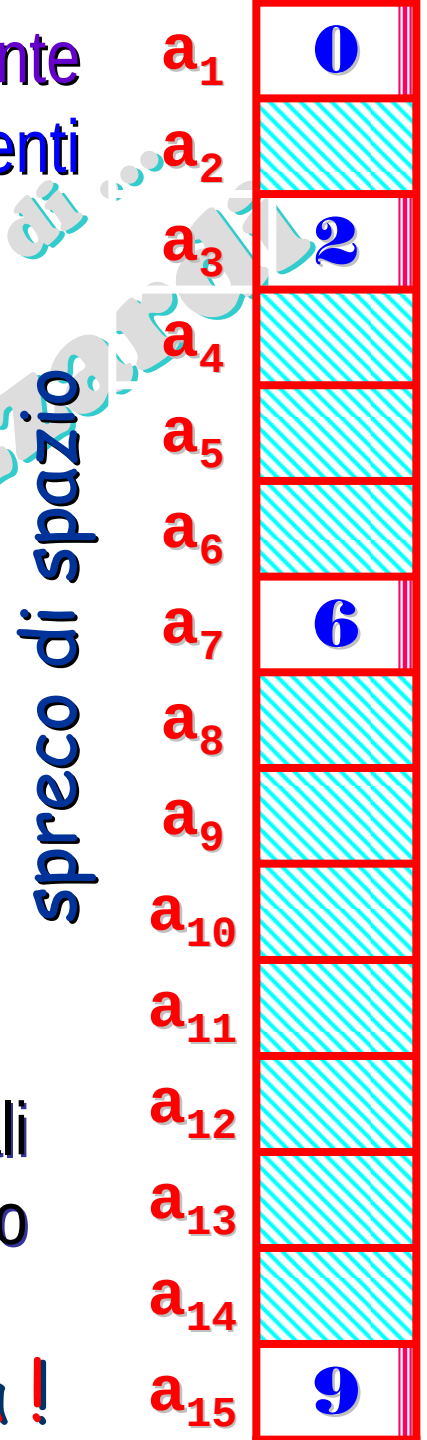
La rappresentazione di un albero binario mediante array va bene solo se l'albero è bilanciato, altrimenti c'è spreco di spazio.



albero degenere
(totalmente sbilanciato)



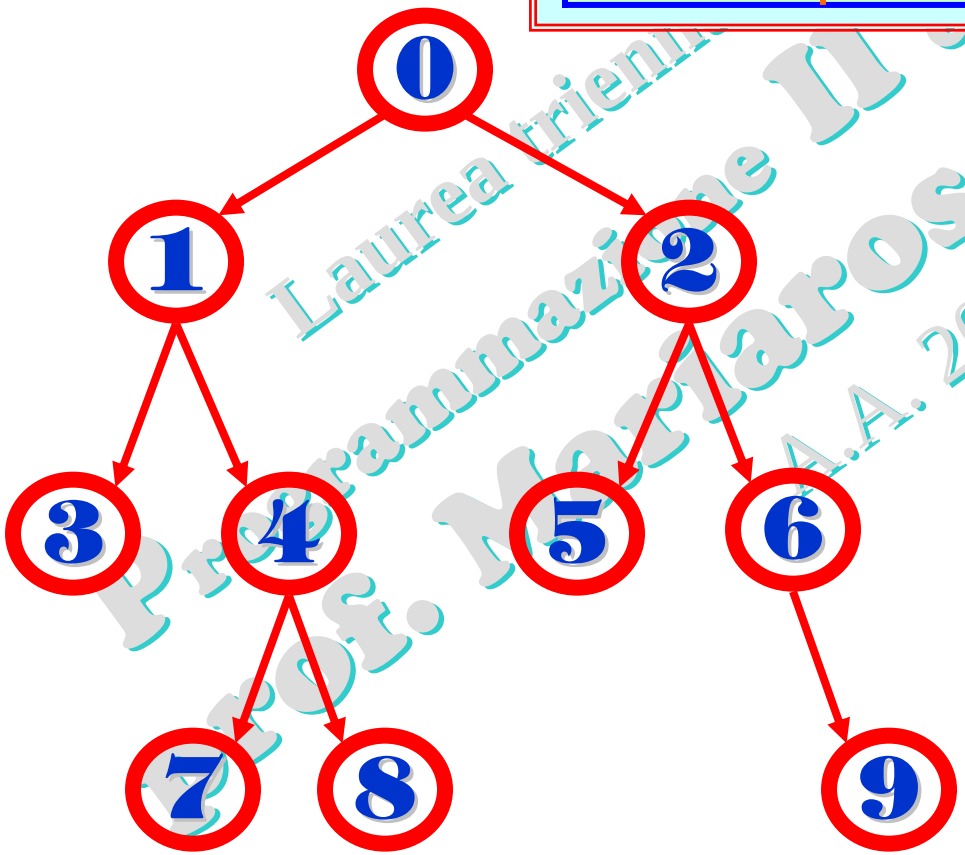
spreco di spazio

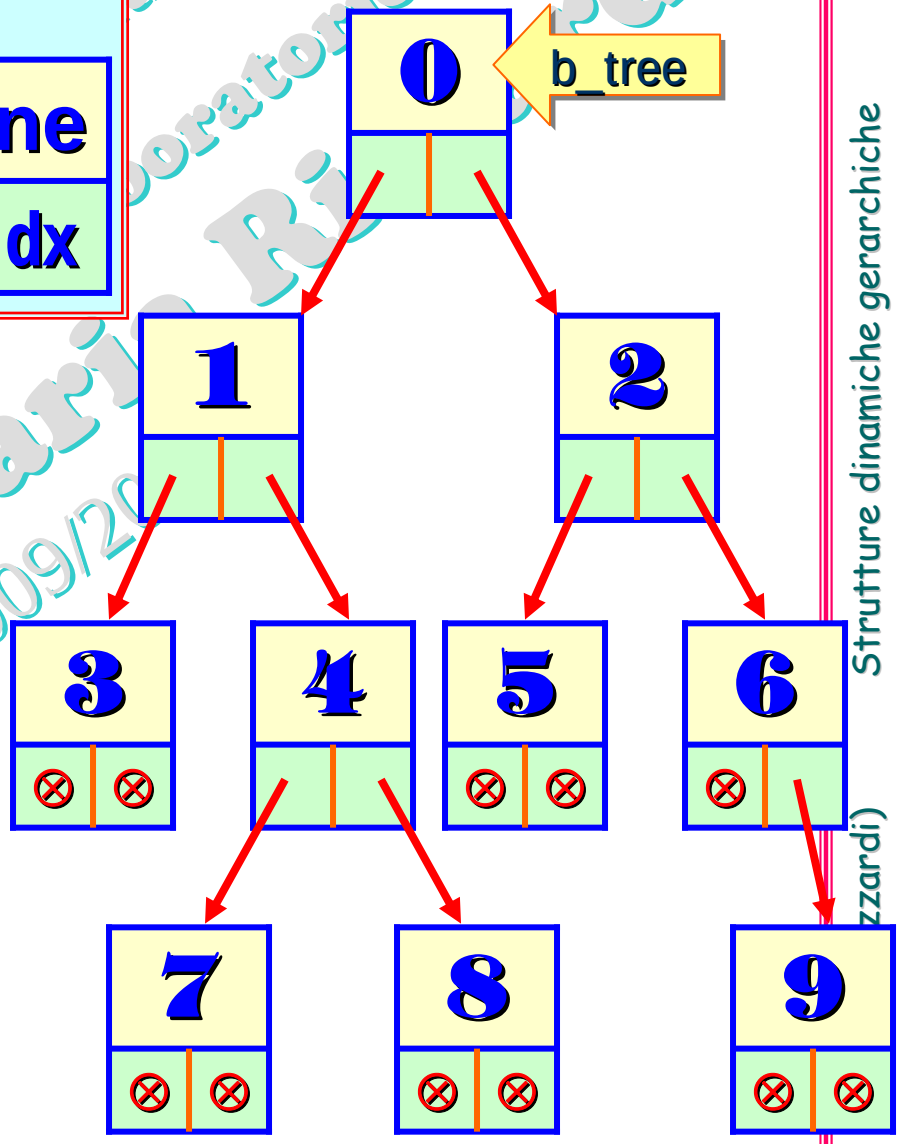


Inoltre ... nell'array sono inefficienti operazioni quali l'inserimento e l'eliminazione di un nodo in quanto richiedono lo spostamento di porzioni di array.

usare la lista !

Esempio: albero binario mediante lista multipla





Qualunque struttura (array o lista) si utilizzi per la memorizzazione di un albero binario, l'algoritmo di visita deve necessariamente far uso di uno

stack

dove conservare l'informazione sull'ordine dei nodi di cui non si è completata la visita dei sottoalberi.

Lo **stack**

va gestito esplicitamente nell' algoritmo iterativo di visita

mentre

è gestito implicitamente dal linguaggio di programmazione se si ricorre ad un algoritmo ricorsivo.

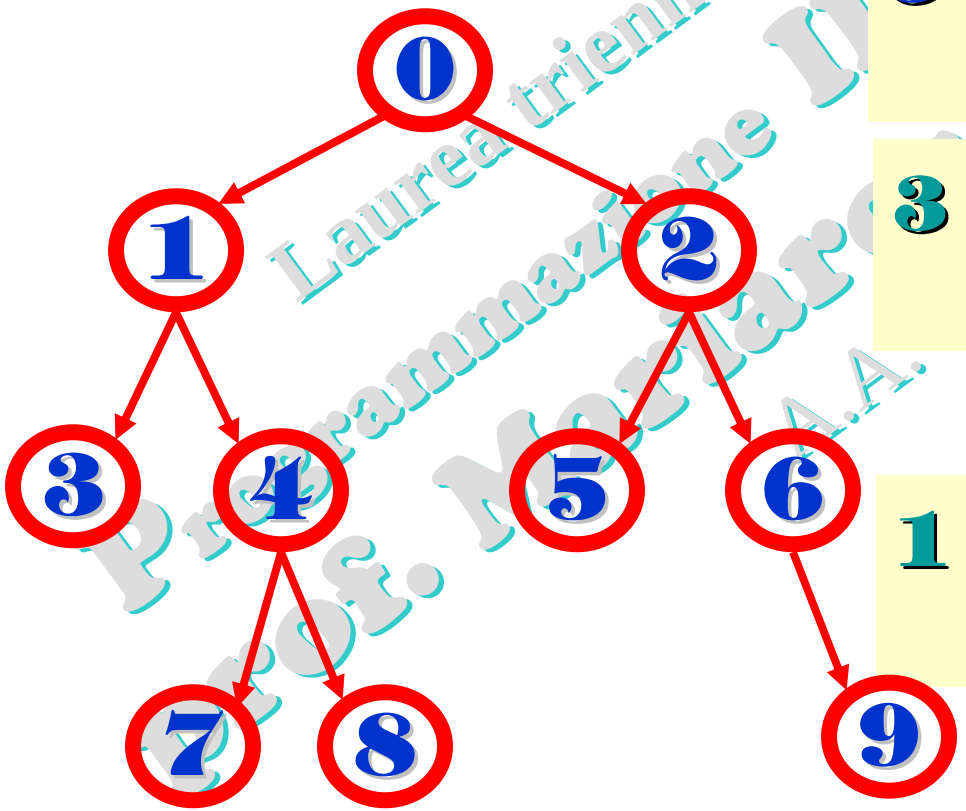
Esempio: Visita preorder con stack

Si attraversa l'albero seguendo la direzione del sottoalbero sinistro di cui la relativa radice è visitata preliminarmente.

Lo stack conserva i nodi di cui non si è ancora visitato il sottoalbero destro.

Esempio: Visita preorder (…continuazione)

Sequenza nodi visitati:



0 → push(**0**)
c'è st.alb. sx (V)

1 → push(**1**)
c'è st.alb. sx (V)

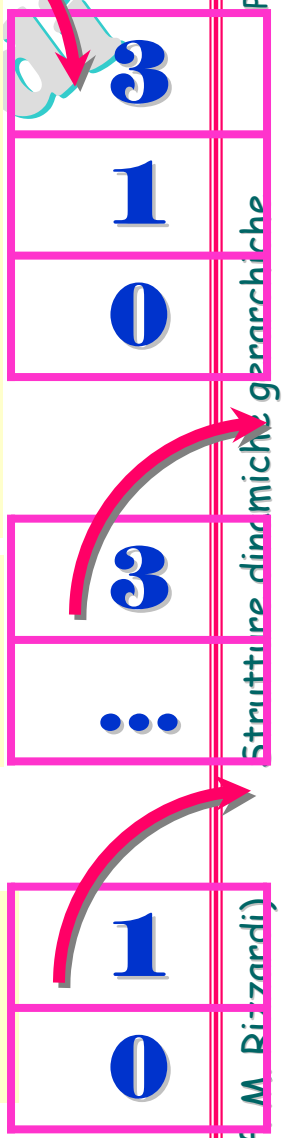
3 → push(**3**)
c'è st.alb. sx (F)

3 ← pop
c'è st.alb. dx (F)

1 ← pop
c'è st.alb. dx (V)

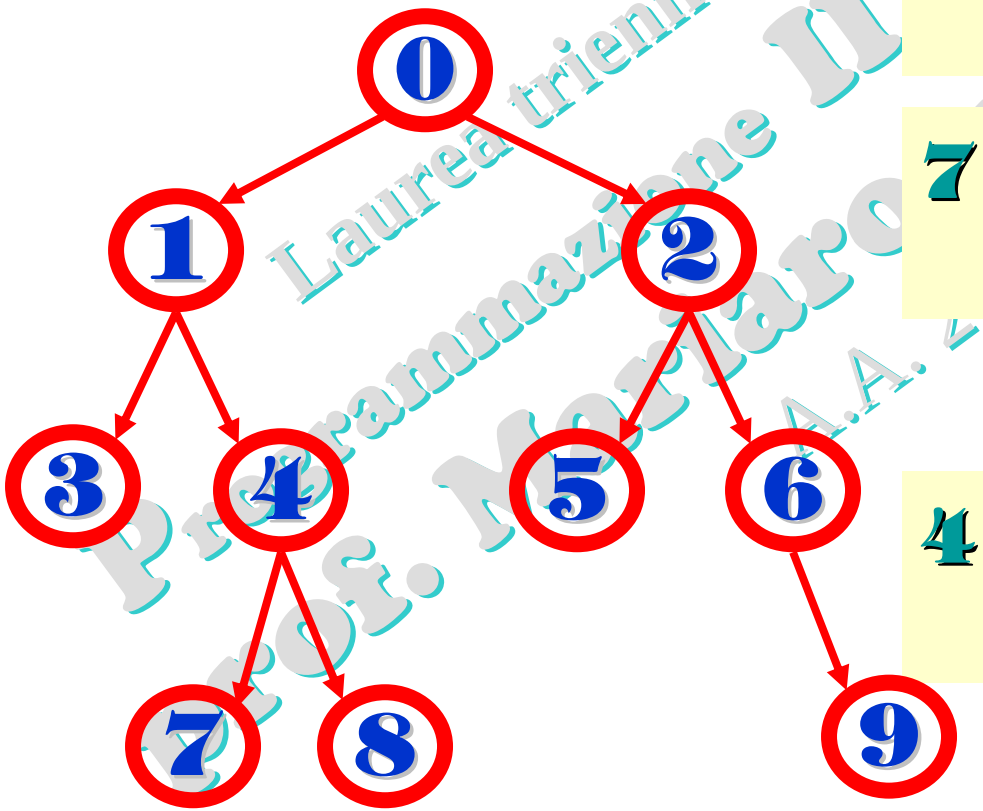
V=vero
F=falso

stack



Esempio: Visita preorder (…continuazione)

Sequenza nodi visitati:



(1) c'è st.alb. dx (V)

4 → push(**4**)

c'è st.alb. sx (V)

7 → push(**7**)

c'è st.alb. sx (F)

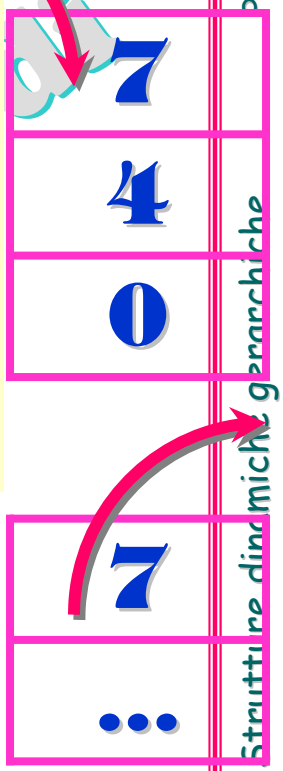
7 ← pop

c'è st.alb. dx (F)

4 ← pop

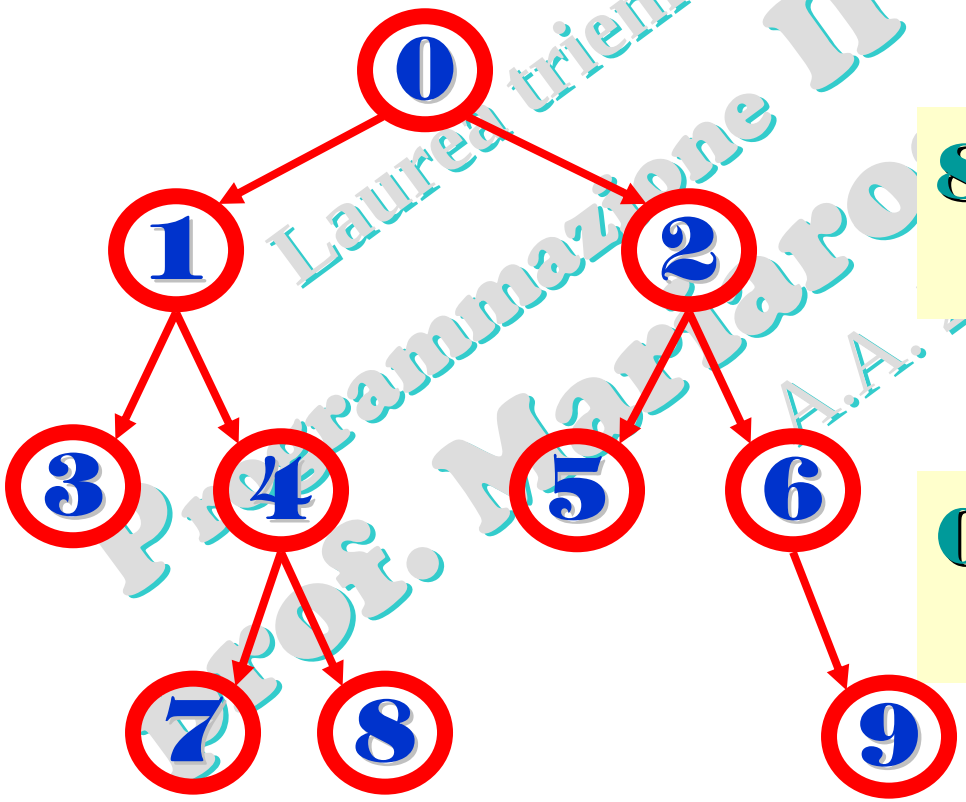
c'è st.alb. dx (V)

stack



Esempio: Visita preorder (…continuazione)

Sequenza nodi visitati:



(4) st.alb. dx (V)

8 → push(**8**)

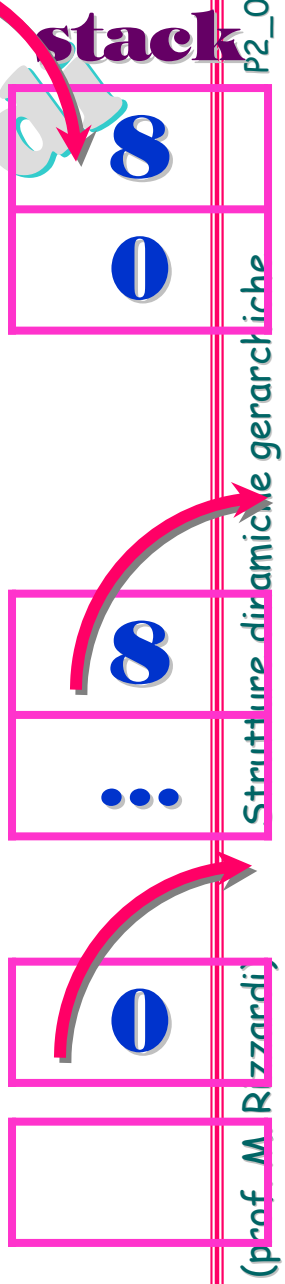
st.alb. sx (F)

8 ← pop

st.alb. dx (F)

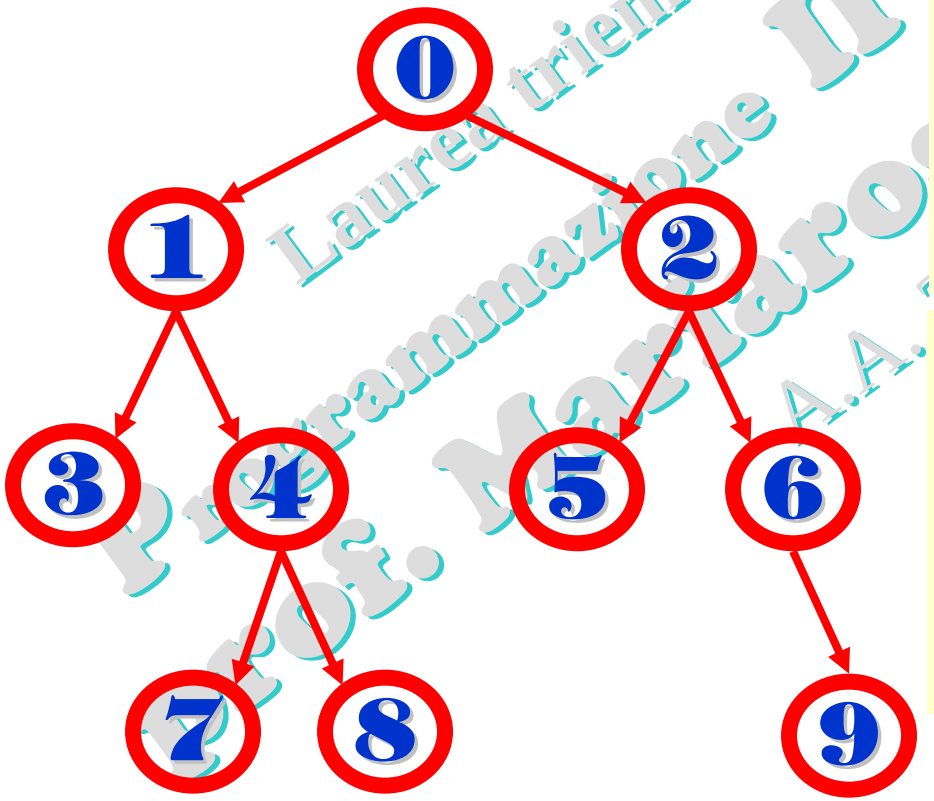
0 ← pop

st.alb. dx (V)



Esempio: Visita preorder (…continuazione)

Sequenza nodi visitati:



(0) st.alb. dx (V)

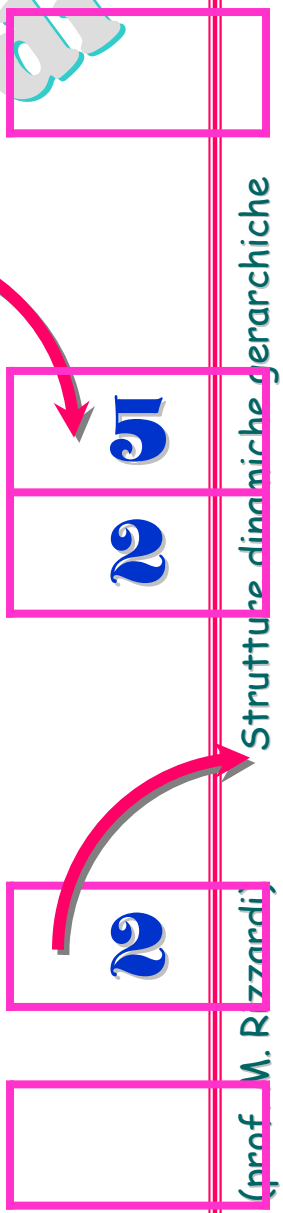
2 → push(2)
st.alb. sx (V)

5 → push(5)
st.alb. sx (F)

5 ← pop
st.alb. dx (F)

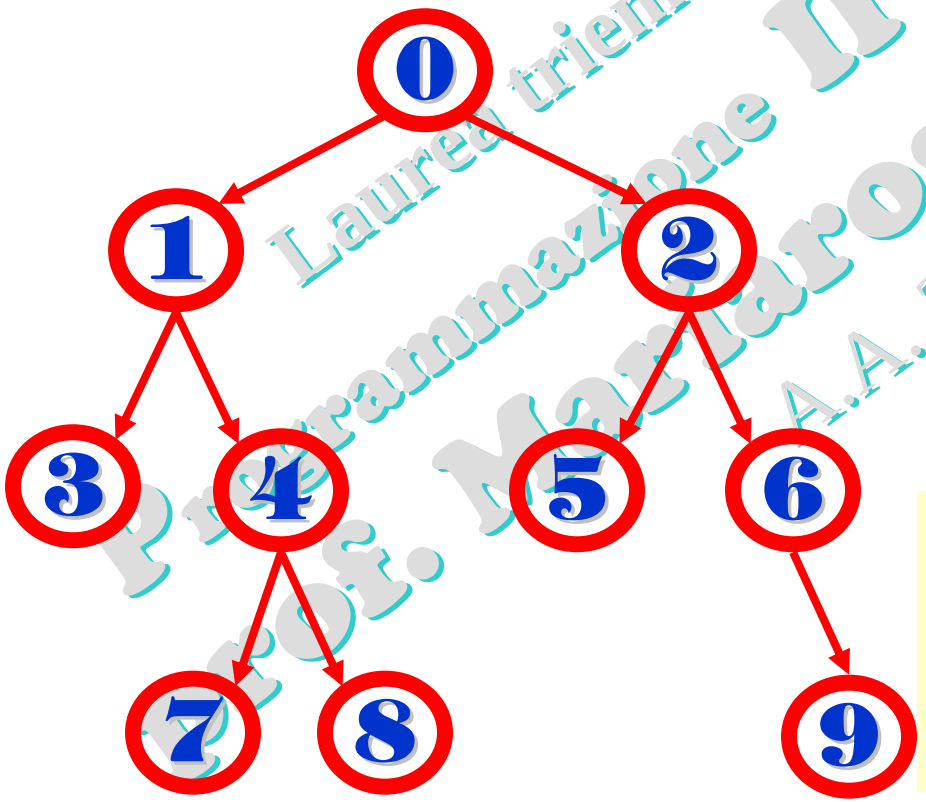
2 ← pop
st.alb. dx (V)

stack



Esempio: Visita preorder (…continuazione)

Sequenza nodi visitati:



(2) st.alb. dx (V)

6 → push(6)

st.alb. sx (F)

6 ← pop

st.alb. dx (V)

9 → push(9)

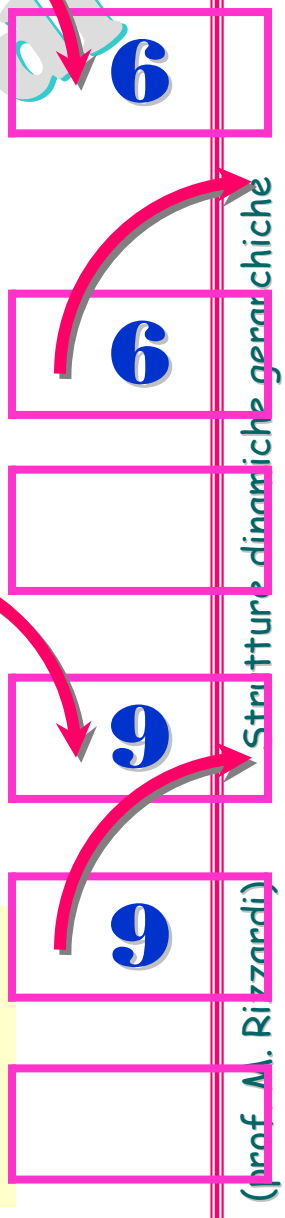
st.alb. sx (F)

9 ← pop

st.alb. dx (F)

pop & stack vuoto=fine

stack

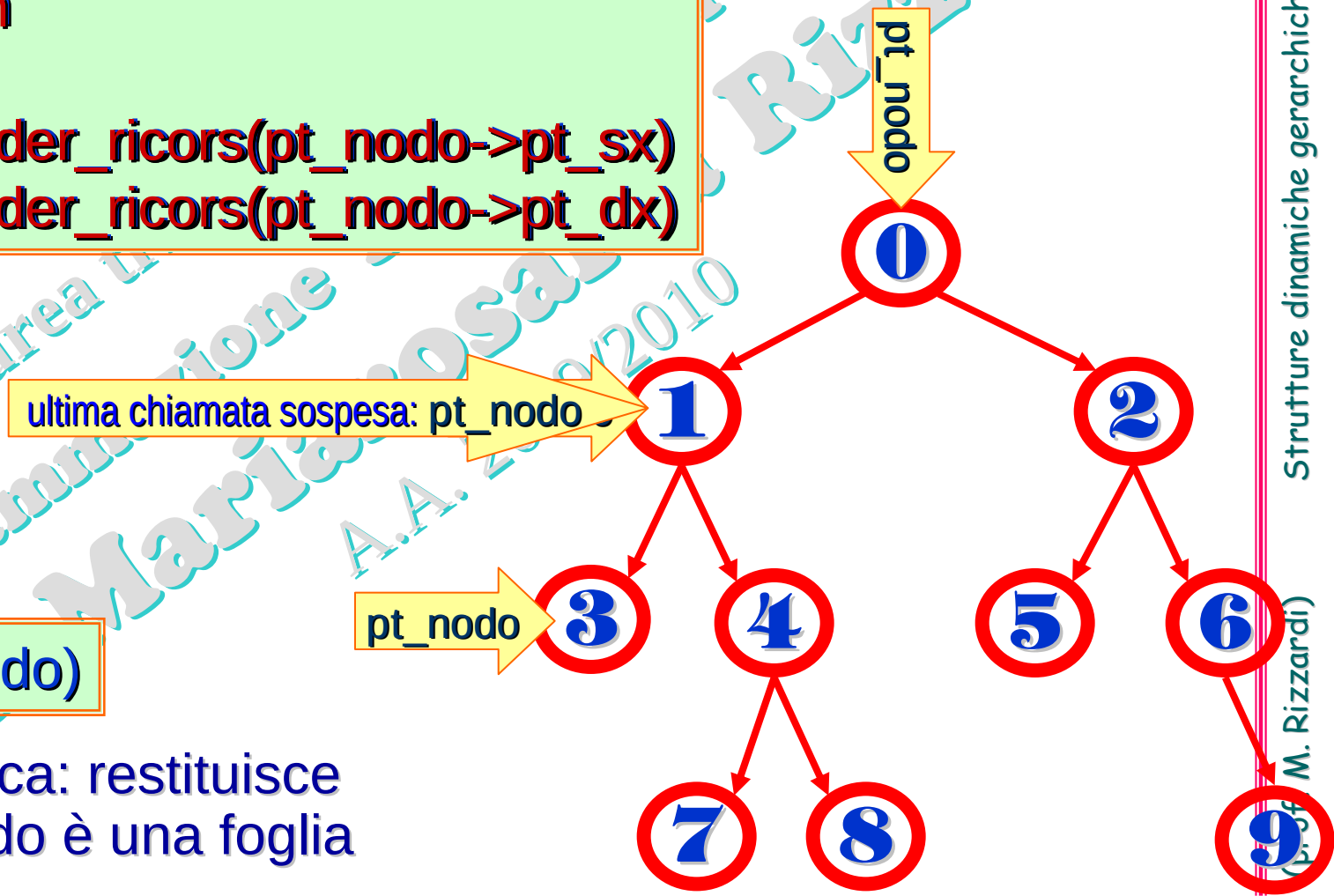


Esempio: Visita preorder ricorsiva

```
preorder_ricors(struct btree *pt_nodo)
visita(pt_nodo->info)
if foglia(pt_nodo)
        return
else
        preorder_ricors(pt_nodo->pt_sx)
        preorder_ricors(pt_nodo->pt_dx)
```

foglia(pt_nodo)

funzione logica: restituisce vero se il nodo è una foglia



Esercizi:

5 Scrivere *function C* per la visita (preorder, inorder e postorder) di un albero binario implementato mediante array.

6 Scrivere *function C* per la costruzione e visita (preorder, inorder, postorder) di un albero binario implementato mediante liste multiple. [liv. 3]