

Lezione 9

Appunti di Davide Scarlata 2024/2025


 Prof: Michele Garetto

 Mail: michele.garetto@unito.it

 Corso:  C

 [Moodle corso C](#)

 [Moodle Lab matricole dispari](#)

 Data: 04/04/2025

Cos'è un albero?

Un **albero** è una struttura dati **non lineare** composta da **nodi**, in cui:

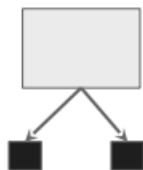
- Ogni nodo può avere **zero o più figli**
- Un albero può essere:
 - **Vuoto**
 - Composto da un **nodo** (radice) e uno o più **sottoalberi**

DEFINIZIONE DI ALBERO

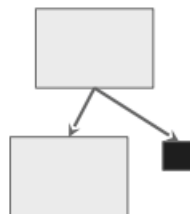
Un albero o è vuoto o è costituito da un nodo che ha come successori uno o più alberi (al più 2 per noi)



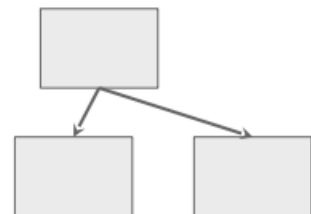
albero
vuoto



nodo che
ha alberi
vuoti come
successori



situazione
mista







nodo che
ha
successori
non vuoti

Terminologia dell'albero

Termine	Definizione
Radice	Nodo di origine, senza antenati
Foglie	Nodi senza successori
Frontiera	Insieme di tutte le foglie
Nodi interni	Nodi con almeno un successore
Discendenti	Nodi che discendono da un nodo dato
Antenati	Tutti i padri di un nodo dato
Arco	Collegamento orientato tra due nodi

Percorsi e profondità

- **Percorso** = sequenza di nodi collegati da **archi**
 -  **Relativo**: tra un nodo e un suo successore
 -  **Absoluto**: dalla **radice** a un nodo (univoco per ogni nodo)
- **Distanza** tra due nodi = numero di archi tra di essi
- **Profondità** = distanza della radice da un nodo
 -  Albero vuoto $\rightarrow -1$
 -  Radice $\rightarrow 0$

Altre definizioni importanti

- **Grado di un nodo**: numero di **figli**
- **Albero di grado N completo**:
 - Tutte le foglie sono alla **stessa profondità N**
 - Tutti i nodi interni hanno **esattamente N figli**
- **Albero degenere**:
 - Ogni nodo ha **al massimo un figlio** (\approx lista)
- **Albero ordinato**:
 - I nodi contengono valori **ordinabili**

- La disposizione dei nodi **rispetta una relazione d'ordine**

ALBERO ORDINATO: ESEMPIO

★ **valori:** numeri interi

★ **relazione d'ordine:** > (MAGGIORE)

★ **proprietà:**

\forall n nodo con figli dell'albero,

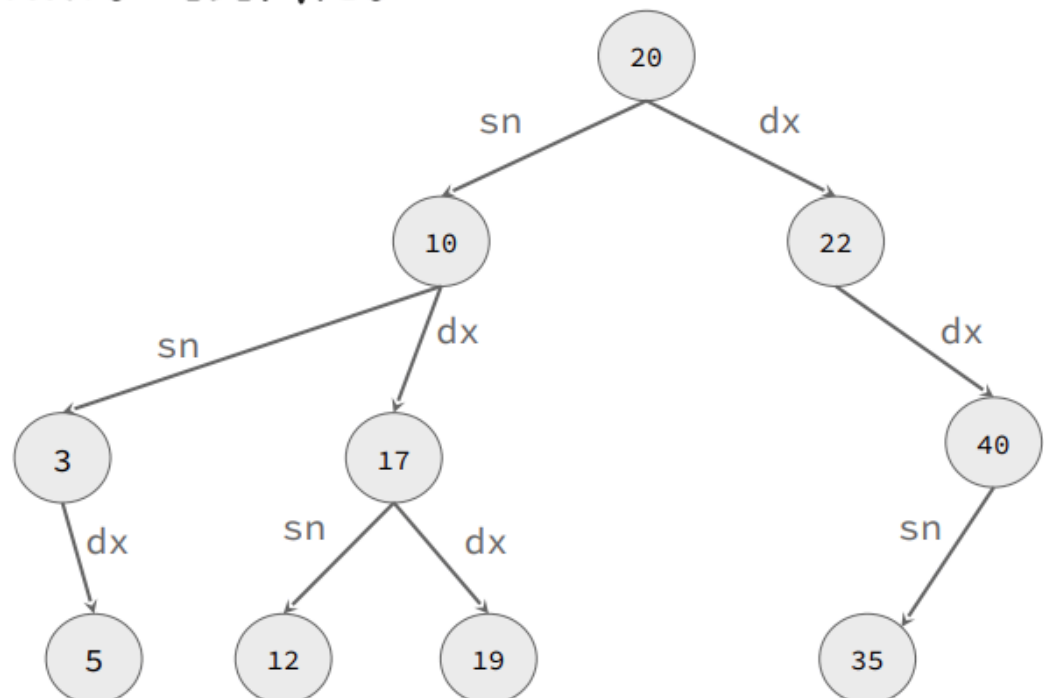
\forall n_sn nodo \in sottoalbero sinistro di n:

$\text{valore}(s_n) < \text{valore}(n)$

\forall n_dx nodo \in sottoalbero destro di n:

$\text{valore}(n) < \text{valore}(n_dx)$

ALBERO ORDINATO: ESEMPIO





Esempio: struttura di albero binario in C

```
typedef struct nodo *tree;

struct nodo {
    int dato1;          // valore contenuto nel nodo
    // altri dati utili (es. etichette, info, ecc.)
    tree left;          // puntatore al sottoalbero sinistro
    tree right;         // puntatore al sottoalbero destro
};
```

operazioni sugli alberi

```
void init(tree *t){// inizializza l'albero
    *t = NULL;
}

tree crea_nodo (int el) {
    tree nuovo = (tree)malloc(sizeof(struct nodo);
    nuovo->dato1 = el;
    nuovo->left = NULL;
    nuovo->right = NULL;
    return nuovo;
}
```

come percorrere un albero (La visita)

- strategie:
- visita in profondità (depth-first visit/search): visita per primo uno dei nodi più distanti dalla radice; realizzata tramite ricorsione o stack di appoggio
- visita in ampiezza (breadth-first visit/search): esplora l'albero per livelli; realizzata tramite coda FIFO di appoggio

visita in profondità

può essere fatta in maniera ricorsiva (con backtracking)
o in maniera iterativa(senza backtracking))

visita ricorsiva

visita in preorder:

```
visita nodo  
visita figlio sinistro  
visita figlio destro
```

stampa prima tutti i nodi prima quelli a sinistra e poi quelli a destra
visita inorder:

```
visita figlio sinistro  
visita nodo  
visita figlio destro
```

percorre l'albero a sinistra finchè ci sono figli sinistri stampa il nodo poi fa il richiamo ricorsivo a destra

visita postorder:

```
visita figlio sinistro  
visita figlio destro  
visita nodo
```

prima eseguiamo il richiamo ricorsivo a sinistra e a destra e poi stampiamo il nodo

implementazioni

stampa in preorder

```
void stampa_preorder(tree t){  
    if(t){  
        printf("%d ", t->dato);  
        stampa_preorder(t->left);  
        stampa_preorder(t->right);  
    }  
    else  
        printf("Albero vuoto");  
}
```

stampa in inorder

```
void stampa_inorder(tree t){  
    if(t){  
        stampa_inorder(t->left);  
        printf("%d ", t->dato);  
    }
```

```

        stampa_inorder(t->right);
    }
    else
        printf("Albero vuoto");
}

```

stampa in postorder

```

void stampa_postorder(tree t){
    if(t){
        stampa_postorder(t->left);
        stampa_postorder(t->right);
        printf("%d ", t->dato);
    }
    else
        printf("Albero vuoto");
}

```

visita in profondità iterativa (con stack)

```

void visita(tree t){
    stack s;
    push(&s, radice);
    while(!Empty(s)){
        current = pop(s);
        //se metto la print qua sarà visita in preorder
        if(current->left){
            push(s, current->left);
        }
        //visita inorder
        if(current->right){
            push(s, current->right);
        }
        //visita postorder
    }
}

```

Visita in ampiezza

viene gestita in maniera iterativa attraverso una coda

allocazione della coda vuota

```
enqueue(coda,radice);  
while(!empty(coda)){  
    current = dequeue(coda)  
    if(current->left){  
        enqueue(coda,nodo sinistro);  
    }  
    if(current->right){  
        enqueue(coda,nodo destro);  
    }  
}
```