5 Alberi

5.0.1 Definizioni

Definizione 5.1 (Albero radicato – *rooted tree*)**.** *Un albero consiste di un insieme di nomi e un insieme di archi orientati che connettono coppie di nodi, con le seguenti proprietà:*

- *un nodo dell'albero è designato come nodo radice;*
- *ogni nodo n, a parte la radice, ha esattamente un arco entrante;*
- *esiste un cammino unico dalla radice ad ogni nodo;*
- *l'albero è connesso.*

Definizione 5.2 (Albero radicato, definizione ricorsiva)**.** *Un albero è dato da:*

- *un insieme vuoto, oppure*
- *una radice e zero o più sottoalberi, ognuno dei quali è albero; la radice è connessa alla radice di ogni sottoalbero con un arco orientato.*

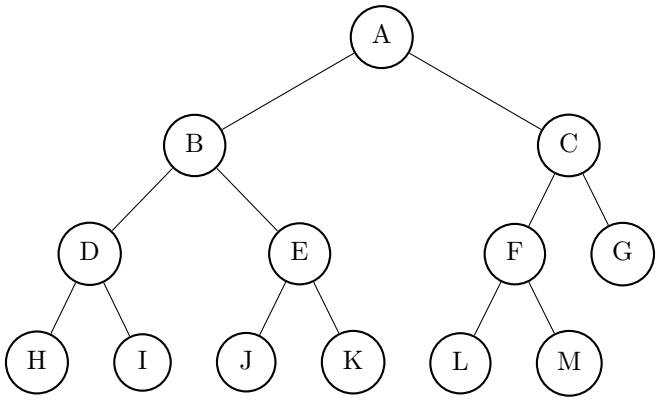
Definizione 5.3 (Profondità – *depth*)**.** *La lunghezza del cammino semplice dalla radice al nodo (misurato in archi).*

Definizione 5.4 (Livello – *level*)**.** *L'insieme dei nodi alla stessa profondità.*

Definizione 5.5 (Altezza dell'albero – *height*)**.** *La profondità massima delle sue foglie.*

5.1 Terminologia

- A è la radice (*root*);
- B, C sono radici dei sottoalberi (*roots of their subtrees*);
- D, E sono fratelli (*siblings*);
- D, E sono figli (*children*) di B;
- B è il padre (*parent*) di D, E;
- H, I, J, K, L, M, G sono foglie (*leafs*);
- gli altri nodi sono nodi interni (*internal nodes*);
- E è lo zio di I;
- B è il nonno di I;
- A è il bis-nonno di I.



5.2 Alberi binari

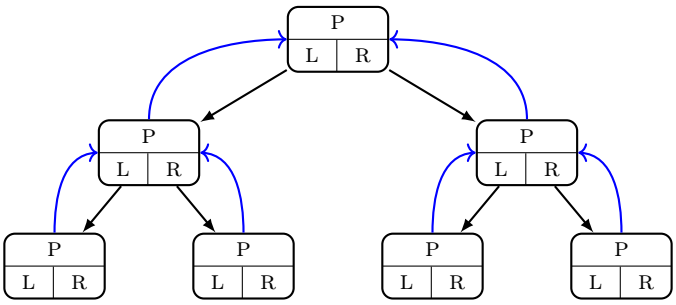
Definizione 5.6 (Albero binario)**.** *Un albero binario è un albero radicato in cui ogni nodo ha al massimo due figli, che vengono identificati come figlio sinistro e figlio destro.*

Nota. *Due alberi T e U che hanno gli stessi nodi, gli stessi figli per ogni nodo e la stessa radice, sono distinti qualora un nodo u sia designato come figlio sinistro di un nodo v in T e come figlio destro del medesimo nodo in U. In altre parole, anche se due alberi hanno lo stesso numero di nodi ed ognuno di questi nodi ha lo stesso numero di figli non è che detto che l'albero risultante sia identico.*

Specifica BINARY TREE

```
// GESTIONE ALBERO

Tree(ITEM v)  // costruisce un nuovo nodo, contenente v, senza figli o genitori
ITEM read  // legge il valore memorizzato nel nodo
write(ITEM v)  // modifica il valore memorizzato nel nodo
TREE parent  // restituisce il padre, oppure nil se questo nodo è radice

// GESTIONE STRUTTURA

// restituiscono il figlio sinistro (destro) di questo nodo,
// restituisce nil se assente
TREE left
TREE right

// inserisce il sottoalbero radicato in t
// come figlio sinistro (destro) di questo nodo
insertLeft(TREE t)
insertRight(TREE t)

// distrugge (ricorsivamente) il figlio sinistro (destro) di questo nodo
deleteLeft
deleteRight
```

5.2.1 Memorizzazione di un albero binario



Vengono memorizzati i seguenti campi:

- *parent*: riferimento al nodo padre;
- *left*: riferimento al figlio sinistro;
- *right*: riferimento al figlio destro.

Uno qualunque di questi oggetti potrebbe essere pari a **nil**, stando ad indicare che sotto di sé non esiste nessun sottoalbero.

5.2.2 Implementazione

Implementazione BINARY TREE in pseudocodice

```
test  // commento
```

5.2.3 Visite

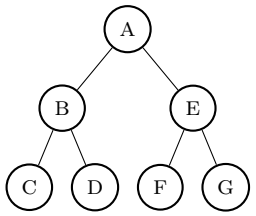
La visita di un albero (o la ricerca) è una strategia per passare attraverso (visitare) tutti i nodi di un albero. Si possono distinguere due tipi di visite:

1. visita in profondità: chiamata anche *Deep-First Search* (DFS), per visitare un albero visita ricorsivamente ognuno dei suoi sottoalberi; esistono tre varianti in base a quando il nodo viene visitato (in pre, in o post order); questa particolare visita richiede il meccanismo di una pila (*stack*);

2. visita in ampiezza: chiamata anche *Breadth First Search* (BFS), per visitare un albero visita ogni livello, uno dopo l'altro partendo dalla radice; richiede il meccanismo di una coda (*queue*).

A seconda di dove scrivo il codice in questo schema ottengo una visita diversa.

```
dfs-schema(TREE t)
    se t ≠ nil allora
        // pre-order visit
        stampa t
        dfs(t.left)
        // in-order visit
        stampa t
        dfs(t.right)
        // post-order visit
        stampa t
```



pre-visita	ABCDEFG
in-visita	CBDAFEG
post-visita	CDBFGEA

5.2.4 Applicazioni

In genere post-visita e in-visita sono quelle più applicate, la pre-visita meno.

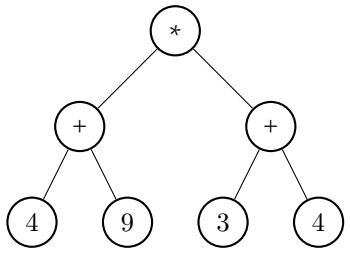
5.2.5 Visita in post-ordine

Una possibile applicazione della visita post-ordine è quella di effettuare un conteggio dei nodi presenti nell'albero.

```
count(TREE t)
    se t == nil allora
        ritorna 0
    allora
        C_ℓ = count(t.left)
        C_r = count(t.right)
        ritorna C_ℓ + C_r + 1
```

5.2.6 Visita in ordine (in-visita)

```
int stampaEspressioni(TREE t)
    se t.left == nil and t.right == nil allora
        // siamo in una foglia
        stampa t.read
    allora
        // sono su un nodo interno
        stampa "("
        stampaEspressioni(t.left)
        stampa t.read
        stampaEspressioni(t.right)
        stampa ")"
```



Stampa: `( 4 + 9 ) * ( 3 + 4 )`