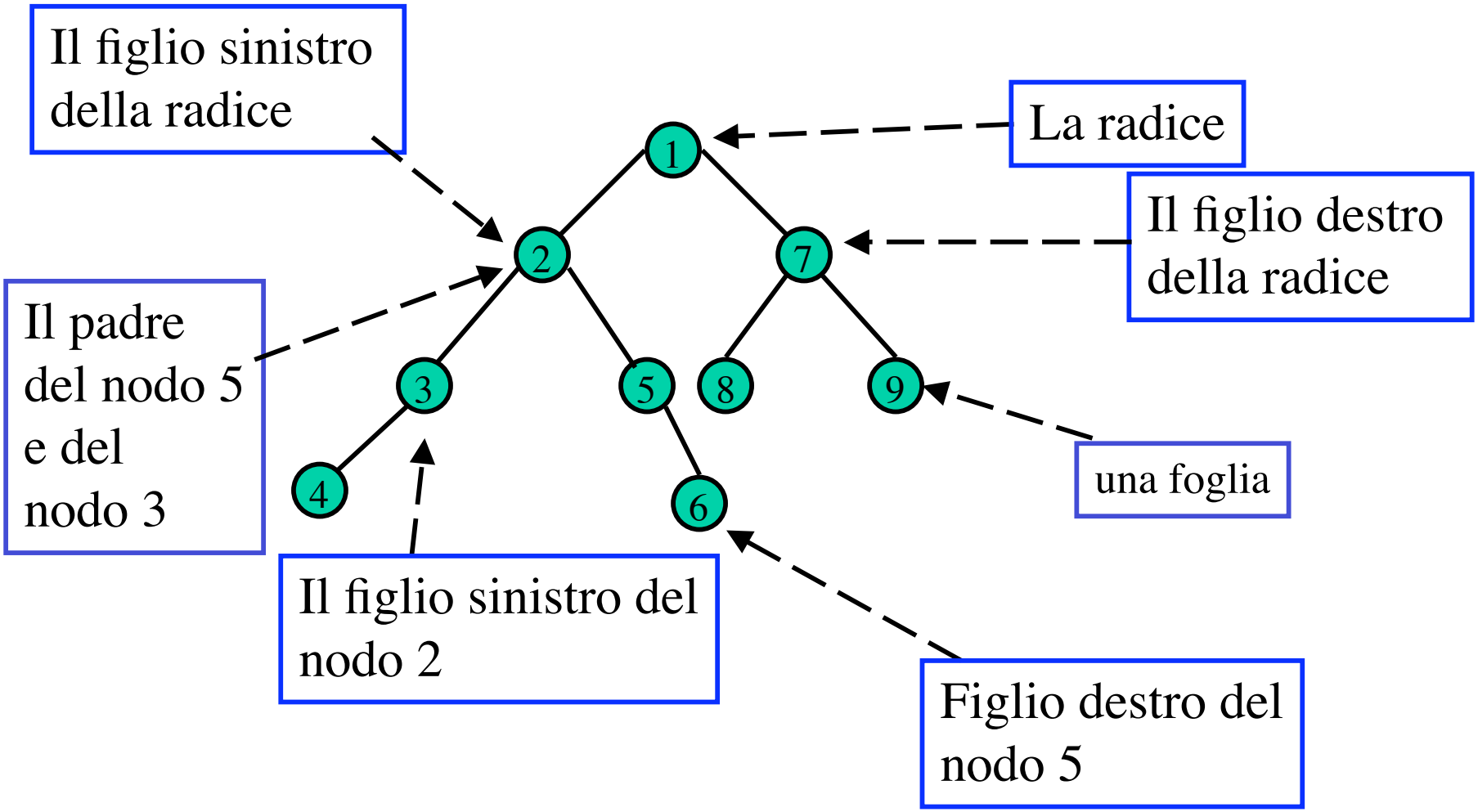
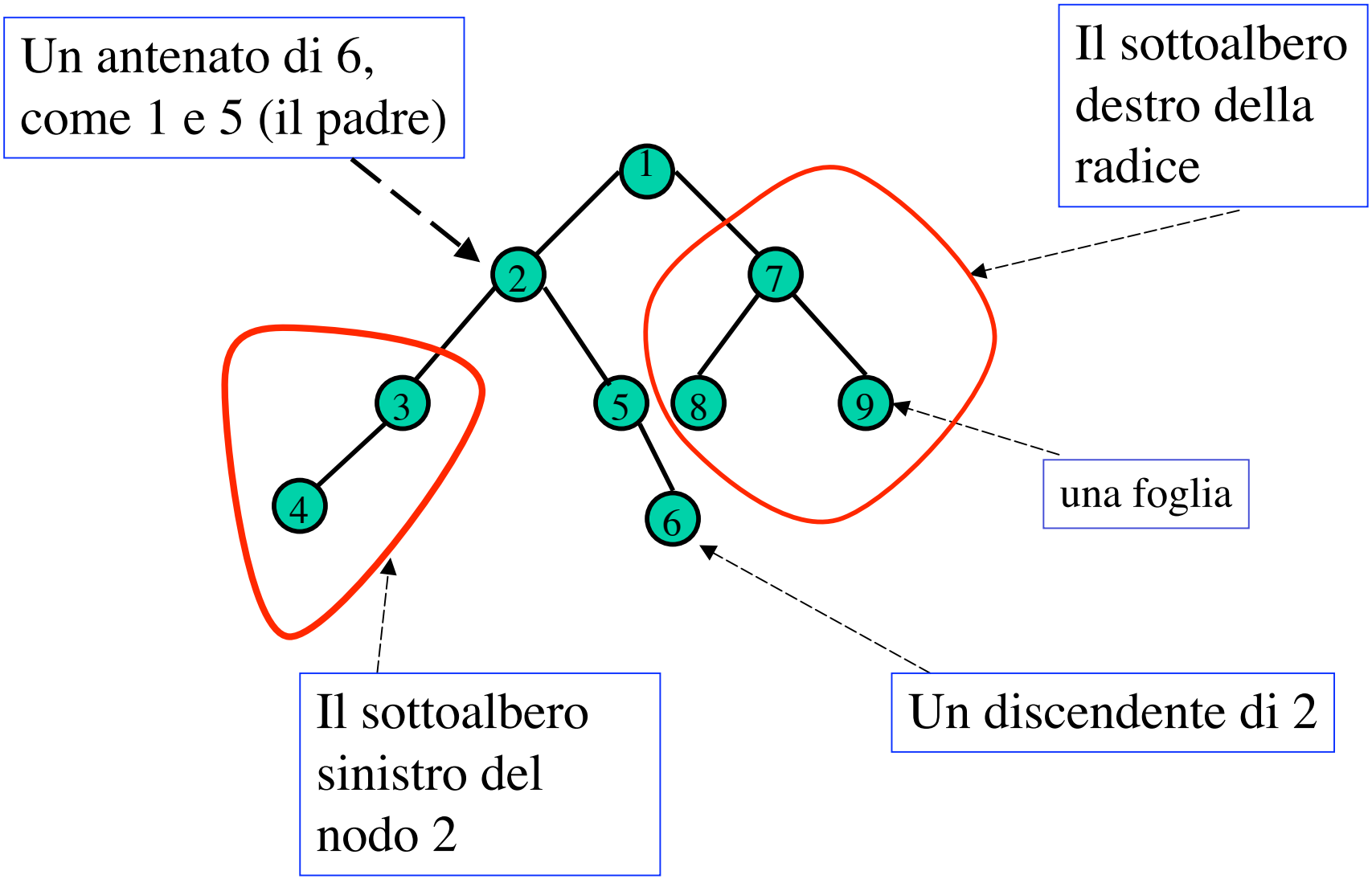
Alberi binari (radicati e ordinati)



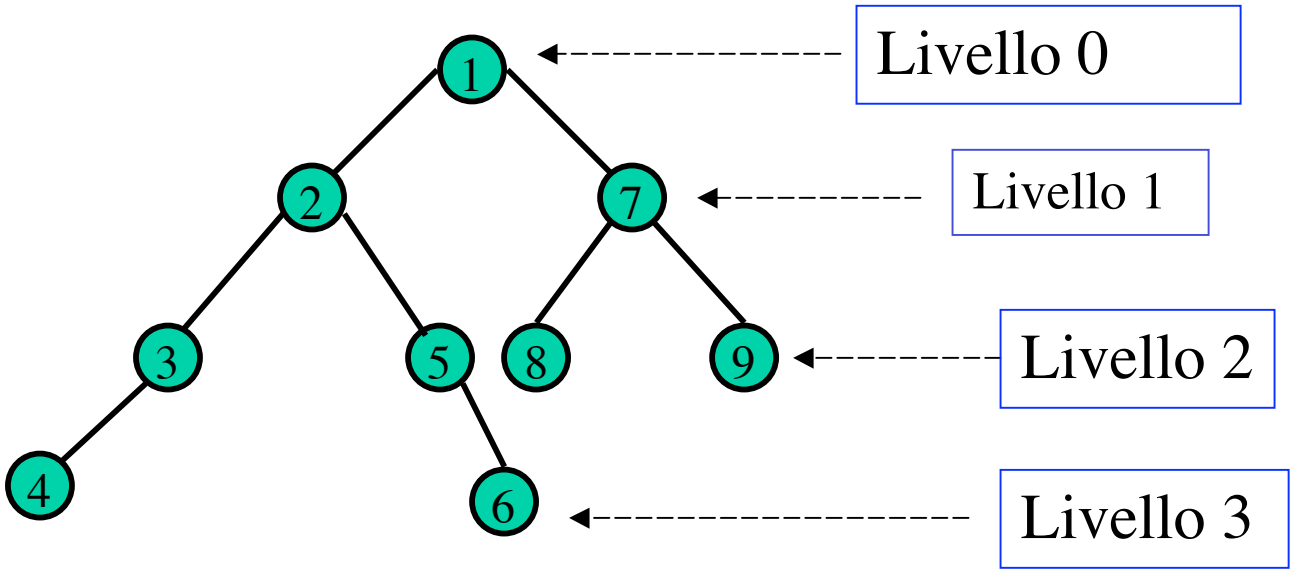
Definizione di albero binario (radicato e ordinato)

Un albero binario è un insieme finito di nodi. L'insieme può essere vuoto (l'albero vuoto). Ma se l'insieme non è vuoto, allora soddisfa le seguenti regole:
1. C'è un nodo speciale chiamato radice
2. A ogni nodo possono essere associati fino a due nodi diversi, chiamati figlio sinistro e figlio destro. Se un nodo c è il figlio di un nodo p, allora diciamo che p è padre di c.
3. Ogni nodo, eccetto la radice ha esattamente un padre. La radice non ha padre.
4. Passando da un nodo a suo padre, purchè presente, si raggiunge la radice.

Alberi binari (radicati e ordinati)



Alberi binari

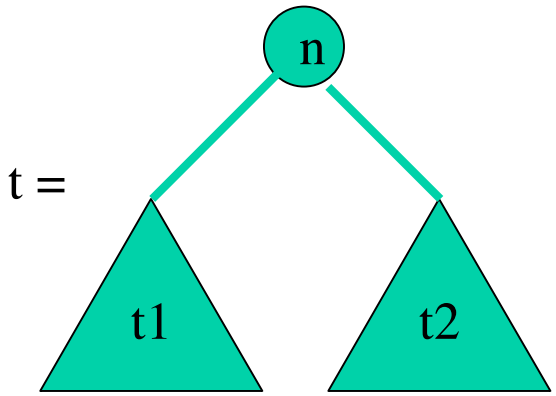


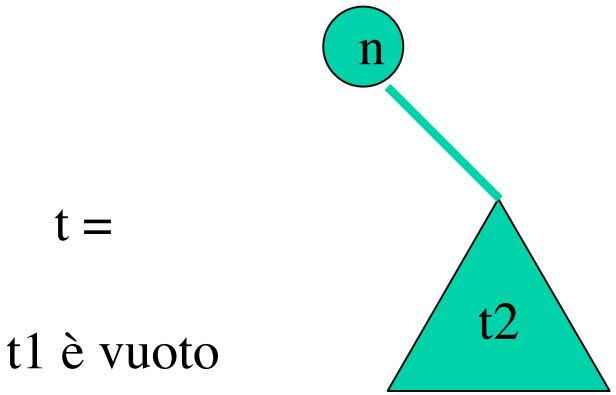
Altezza dell'albero = max livello delle foglie

In alcuni testi livello di un nodo = profondità di un nodo
Altezza dell'albero = profondità dell'albero

Definizione ricorsiva di albero binario:

BASE:L'albero vuoto ∅ è un albero binario
INDUZIONE: Se t1 e t2 sono alberi binari, e n è un nodo, allora t è un albero binario se ottenuto collegando il nodo n ordinatamente alla radice di t1, se t1 non è vuoto altrimenti t non ha sottoalbero sinistro, e collegando n alla radice di t2 se t2 non è vuoto, altrimenti t ha il sottoalbero destro vuoto.

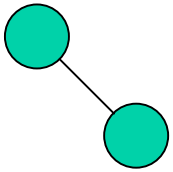




t=

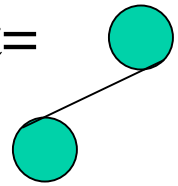
L'albero con un solo nodo, che è anche la sua radice

t=



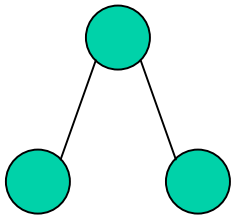
L'albero t costruito prendendo come sottoalberi l'albero vuoto e un albero con un solo nodo

t=



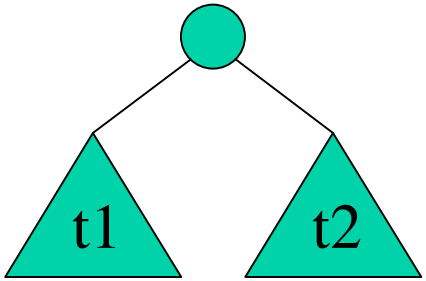
L'albero t simmetrico al precedente

t=

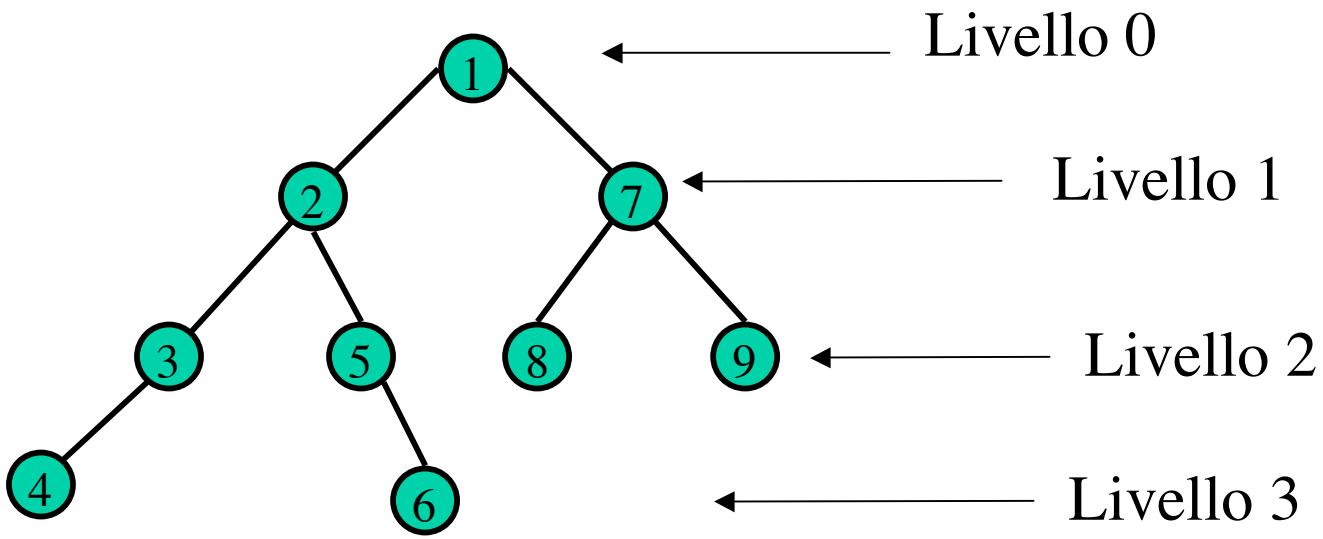


L'albero t costruito prendendo come sottoalberi due alberi con un solo nodo.

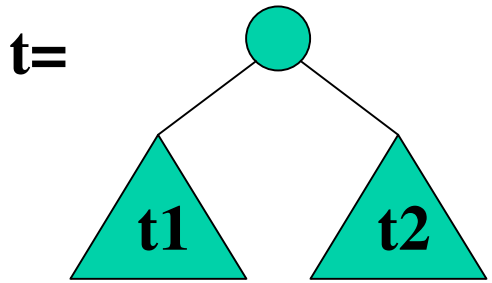
t=



L'albero t costruito prendendo come sottoalberi due generici alberi t1, t2, t1 è il **sottoalbero** sinistro di t e t2 quello destro



Definizione ricorsiva di livello di un nodo:
La radice è a livello 0
Se un nodo è a livello n, i suoi figli sono al livello n+1



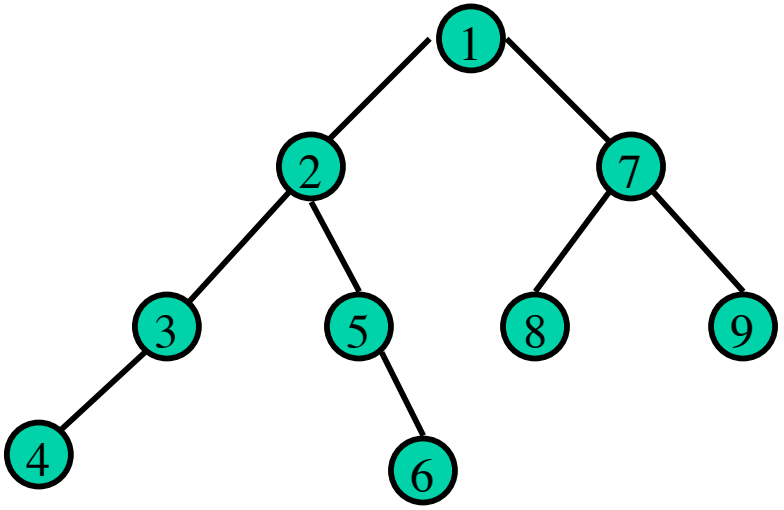
Alt(∅) = -1

Alt(t) = max(Alt(t1),Alt(t2)) + 1

NumNodi(∅) = 0

NumNodi(t) = NumNodi(t1) + NumNodi(t2) + 1

Esercizio: costruire due funzioni C, alt e numNodi, che ricevendo in input la radice di un albero, restituiscano rispettivamente l'altezza e il numero dei nodi dell'albero e una funizxone che dato un albero e un suo nodo restituisca il livello del nodo.



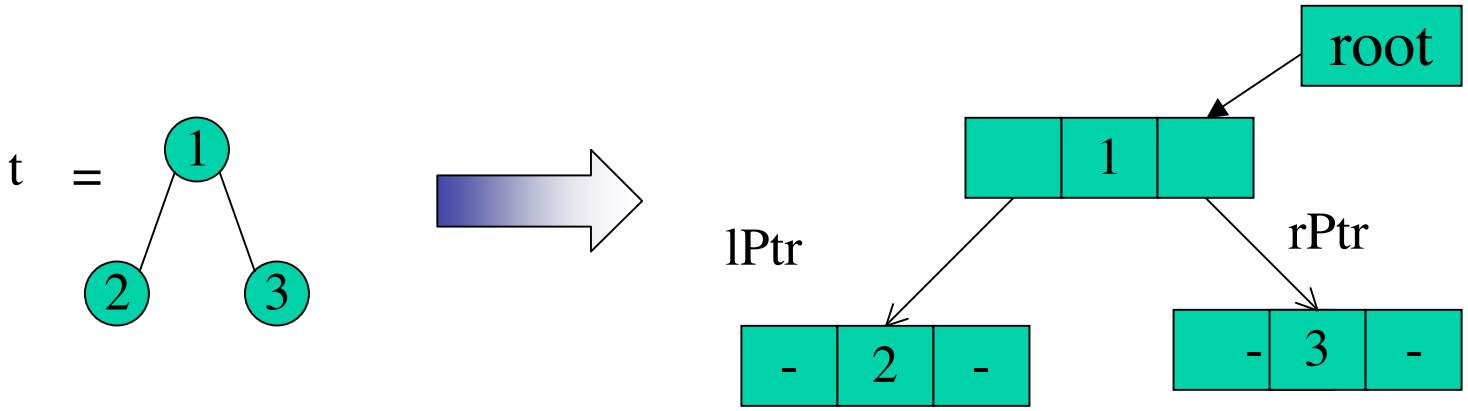
Un cammino da un nodo n a un nodo m è una sequenza di nodi connessi da archi che portano da n a m.

Esempio: cammino da 4 a 6: 43256

La lunghezza di un cammino è il numero di archi che connettono i nodi del cammino (uno di meno del numero dei nodi).
Esempio: il cammino da 4 a 6 ha lunghezza 4.

L'altezza può essere anche definita come la massima lunghezza di un cammino radice-foglia, che contiene solo archi da padre a figlio.

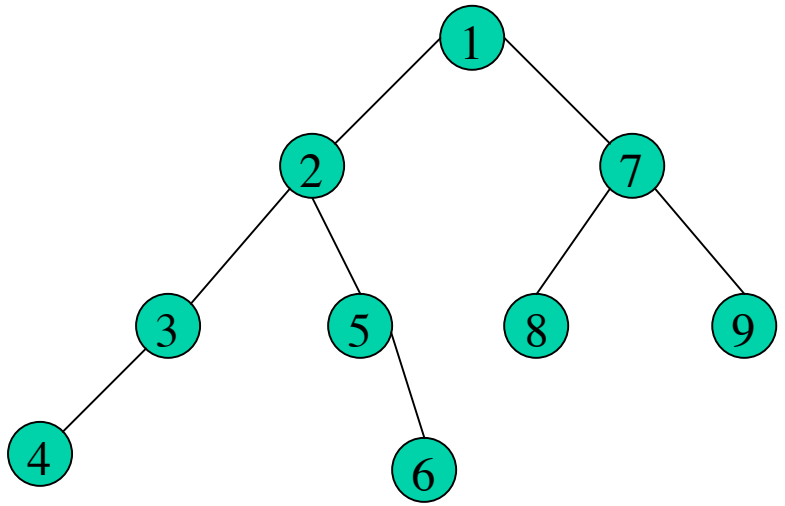
Rappresentazione in memoria di un albero




```
struct tNode {
 struct tNode *lPtr;
 int info;
 struct tNode *rPtr;
   };
typedef struct tNode TreeNode;
typedef TreeNode * TreePtr;
```

Visita inorder: visita il sottoalbero sinistro
visita la radice
visita il sottoalbero destro

Esempio:

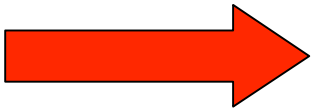


Sequenza nodi visitati : 432561879

```
struct tNode {
struct tNode *lPtr;
int info;
struct tNode *rPtr; };

typedef struct tNode TreeNode;
typedef TreeNode * TreePtr;

void inOrder(TreePtr tPtr)
{
  if (tPtr)
    {
     inOrder(tPtr->lPtr);
     visit(tPtr);
     inOrder(tPtr->rPtr);
     }
}
```



Visita postOrder : visita il sottoalbero sinistro
visita il sottoalbero destro
visita la radice

Visita preOrder : visita la radice
visita il sottoalbero sinistro
visita il sottoalbero destro

Esercizio: Si definisca le funzioni C ricorsive che implementano le visite in postOrder e in preOrder e si definisca una funzione ricorsiva il cui prototipo è
void creaAlbBin(TREEPTR *);
che costruisce un albero binario radicato nel parametro in input chiedendo all'utente di fornire i valori da inserire nei nodi e se tali valori devono essere nel figlio sinistro o destro.

```
void invInOrder(TREEPTR treePtr, int lev)
/* stampa un albero mettendone in evidenza la struttura,*/
/* sfruttando il livello di ogni nodo*/
{int i;
  if (treePtr != NULL) {
    lev ++;
    invInOrder(treePtr->rightPtr,lev);
    for (i=0;i<(5*lev);i++) printf("%s", " ");
    visit(treePtr);
    printf("\n");
    invInOrder(treePtr->leftPtr,lev);
  }
}
```

Per esempio

```
          20
    10
                40
          15
                35
```