Esercitazione 6

Alberi binari di ricerca

Struttura base

- Rappresentabile attraverso una struttura dati concatenata in cui ogni nodo è un oggetto di tipo struttura

- *Ogni nodo contiene*:
  - campo chiave (***key***)
  - (dati satelliti)
  - puntatori ***left***, ***right*** e ***parent*** (eventualmente **NULL**)

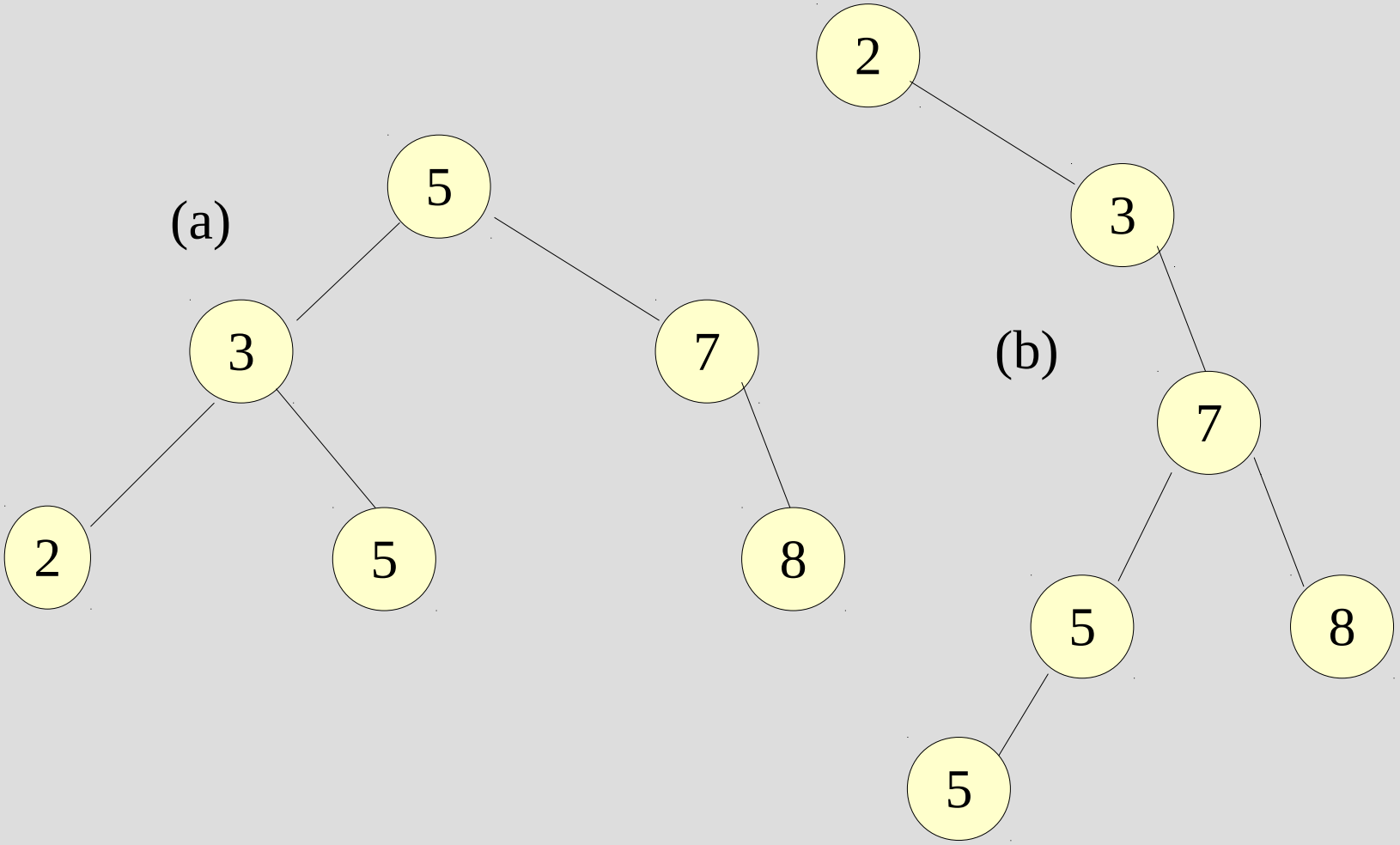
Struttura dati in C++

```
//definizione della struttura nodo
struct elem {
      int key;  //ma può essere != intero...
      elem* left;
      elem* right;
      elem* parent;
    };
//definizione del puntatore a nodo
typedef elem* pnode;
```

Proprietà

- Le chiavi sono memorizzate in modo da soddisfare le ***proprietà degli alberi binari di ricerca***
- Sia **x** un nodo in un albero binario di ricerca:
  - Se **y** è un nodo nel sottoalbero sinistro di **x** allora **y->key <= x->key**
  - Se **y** è un nodo nel sottoalbero destro di **x** allora **y->key >= x->key**

Esempi di alberi binari di ricerca



Operazioni

- Andremo ad implementare alcune fra le operazioni più comuni che si possono effettuare sugli alberi binari di ricerca:
  - ***Scansione completa dell'albero***
  - ***Inserimento di un elemento***
  - ***Ricerca di un elemento in base al valore di chiave***
  - ***Cancellazione di un elemento con un determinato valore di chiave***

Scansione completa dell'albero

- **Problema**: attraversare tutto l'albero senza dimenticare nessun nodo
- Utile per diverse operazioni
- Es. *stampa dell'albero*
  - *Problema: stampare una volta sola ciascun nodo dell'albero*
- **Idea di base**: visitare **ricorsivamente, per ogni nodo,** il sottoalbero sinistro, poi il sottoalbero destro

Scansione completa dell'albero

- Effettuata attraverso ***algoritmi ricorsivi***
  - ***Inorder (attraversamento simmetrico)*** elenca la radice tra i valori del sottoalbero sinistro e quelli del sottoalbero destro
  - ***Preorder (attraversamento anticipato)*** elenca la radice prima dei valori dei suoi sottoalberi sinistro e destro
  - ***Postorder (attraversamento posticipato)*** elenca la radice dopo i valori dei suoi sottoalberi sinistro e destro

Scansione inorder: funzionamento

- Si visita prima la parte sinistra dell'albero, poi quella destra
- ***Per ogni nodo x visitato***:
  - Si scende (chiamata ricorsiva) verso il figlio sinistro
  - Si stampa il valore del nodo **x**
  - Si scende (chiamata ricorsiva) verso il figlio destro
- ***Condizione di terminazione***: chiamata su un nodo inesistente (puntatore **NULL**)

Codice – scansione inorder

- Funzione che stampa tutti gli elementi del sottoalbero che parte dal nodo **p**

```
void print_tree(pnode p) {
    if (p != NULL) {
      print_tree(p->left);
      cout << p->key << endl;
      print_tree(p->right);
    }
 }
```

Inserimento di un elemento

- Inserimento di un elemento con chiave **v** in un albero binario di ricerca **T**
- Ogni inserimento richiede la ***modifica della struttura dati*** dell'albero per mantenerne valide le proprietà
- Inserire elementi è un'operazione relativamente semplice (la cancellazione sarà più complessa...)
  - L'inserimento avviene sempre in un **nodo foglia**

Funzionamento

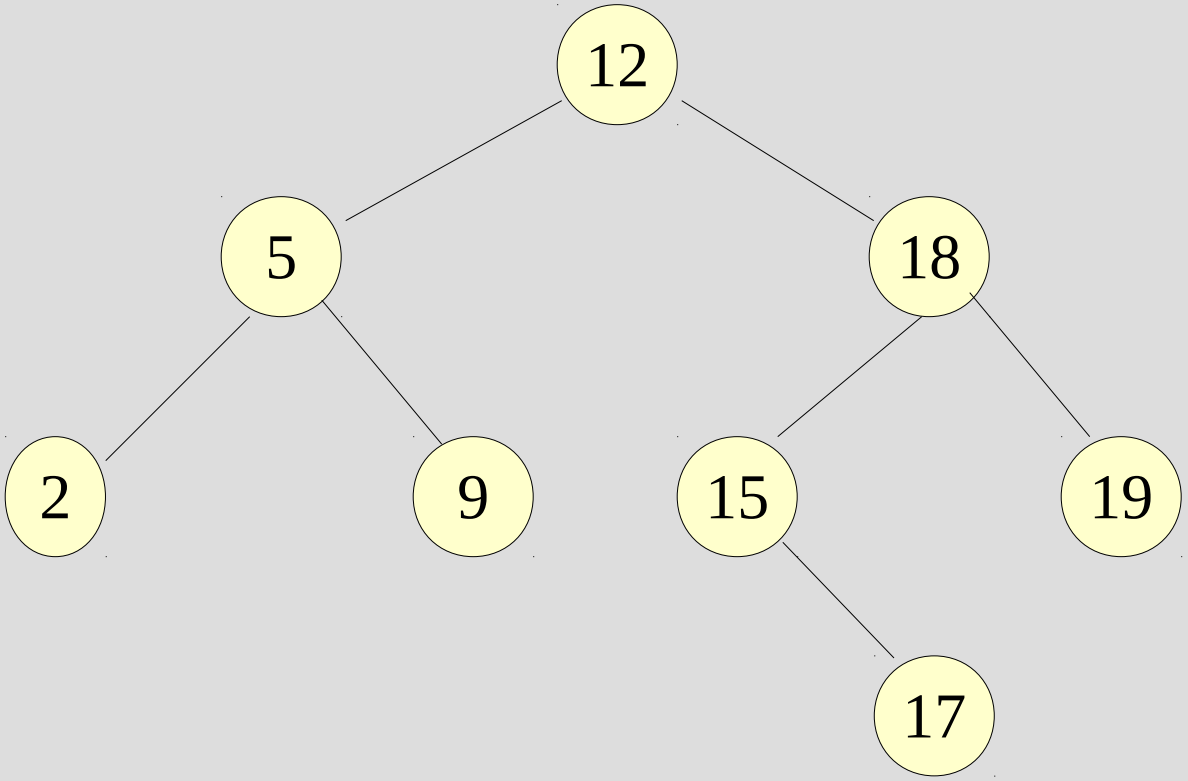
- Procedura **Tree-Insert(T,z)**, che inserisce il nodo **z** nell'albero **T**
- Ricerca della posizione in cui inserire il nuovo nodo
- ***Scansione dell'albero a partire dalla radice verso il basso***
- Utilizzo di ***due puntatori ausiliari***:
  - **x** segue il percorso (nodo corrente)
  - **y** punta al padre di **x**

Percorso di scansione dell'albero

- Il ***percorso di scansione dipende dall'esito dei confronti*** tra ***z->key*** e ***x->key***:
  - Se ***z->key < x->key*** la scansione prosegue nel sottoalbero di sinistra
  - Se ***z->key > x->key*** la scansione prosegue nel sottoalbero di destra
  - La scansione termina quando a ***x*** viene assegnato valore **NULL**
- Al termine della scansione si inserisce il nuovo nodo (*utilizzando il puntatore* ***y***)

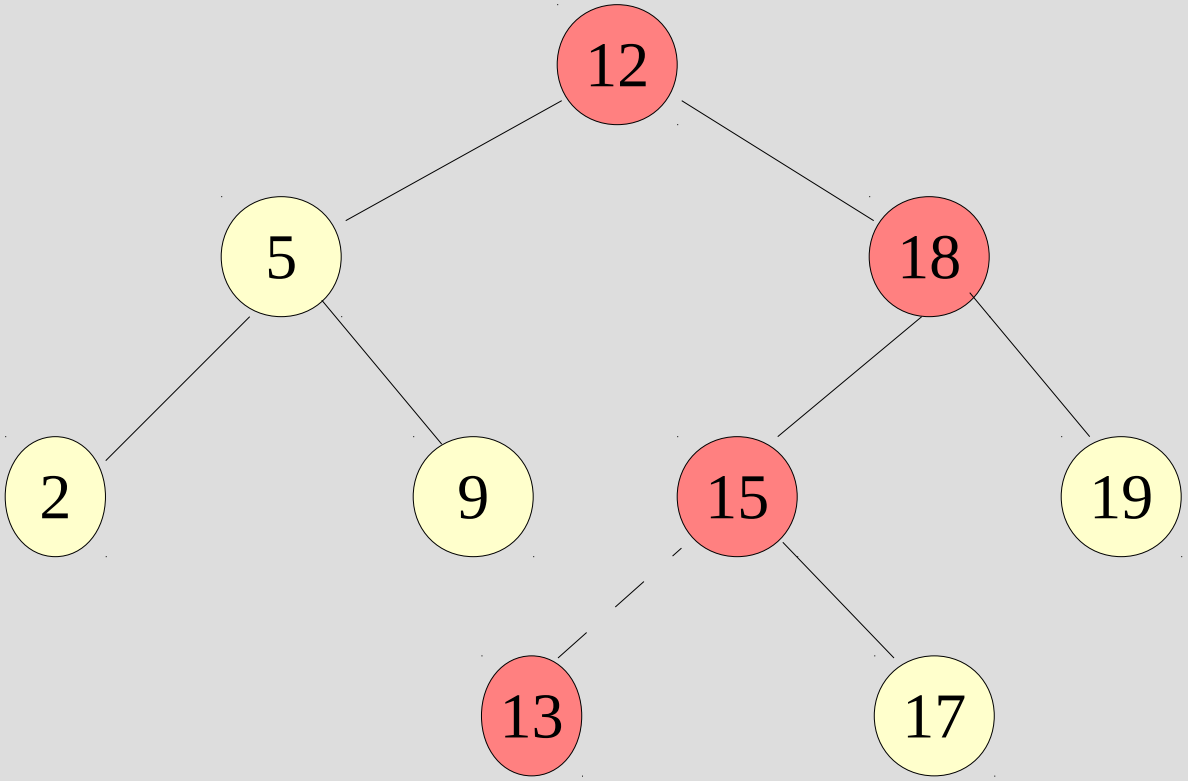
Esempio

- Inserimento del valore 13



Esempio

- Inserimento del valore 13



E quando z->key==x->key?

Programma

- *btree.cc*
- Programma che crea un albero binario di ricerca a partire da valori inseriti da utente
- All'inserimento di un valore pari a 0 il programma termina con la stampa dell'albero (*print_tree*)
- La funzione *print_tree* utilizza un algoritmo di visita simmetrico (*inorder*) dell'albero

Ricerca di un elemento

- Tipica operazione su un albero binario di ricerca: ***ricerca di una chiave k***
- Dati **k** e un puntatore **p** alla radice dell'albero, la procedura **Tree-Search(p,k)** restituisce:
  - un puntatore a un nodo con chiave **k**, se esiste
  - il valore **NULL** altrimenti

Ricerca: funzionamento

- Simile all'inserimento: scansione dell'albero a partire dalla radice verso il basso
- Per ogni nodo x incontrato:
  - Se **k < *x->key*** la ricerca prosegue nel sottoalbero di sinistra
  - Se **k > *x->key*** la ricerca prosegue nel sottoalbero di destra
  - La ricerca termina quando ***k == x->key*** oppure ***x == NULL***

Cancellazione di un elemento

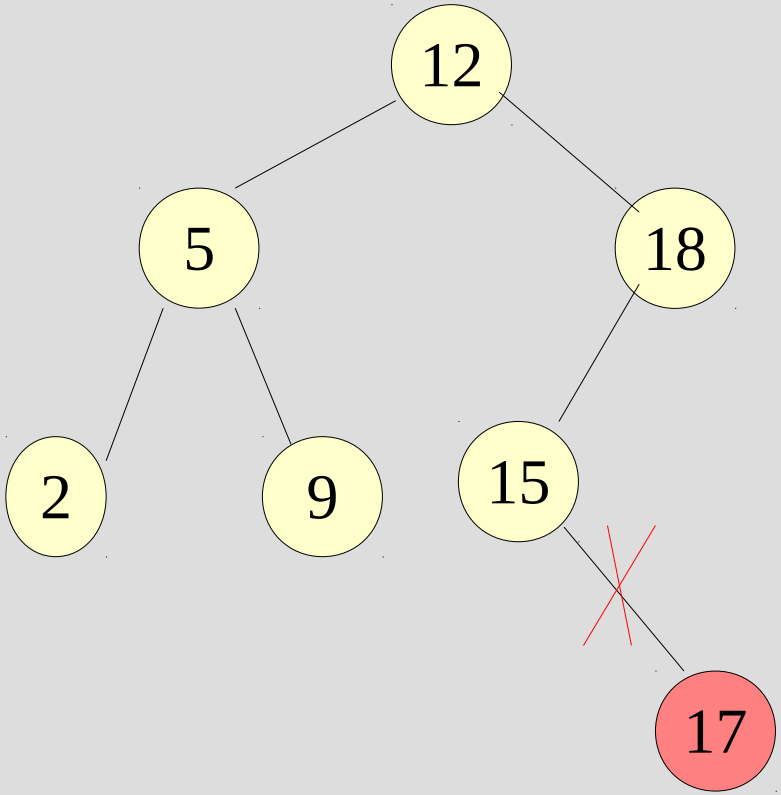
- La ***cancellazione*** di un nodo **z** dell'albero è ***operazione piuttosto complessa***
  - Deve essere mantenuta la struttura dell'albero binario di ricerca
  - Coerenza dei puntatori
- Procedura **Tree-Delete(T, z)** Richiede un puntatore al nodo **z** nell'albero **T**

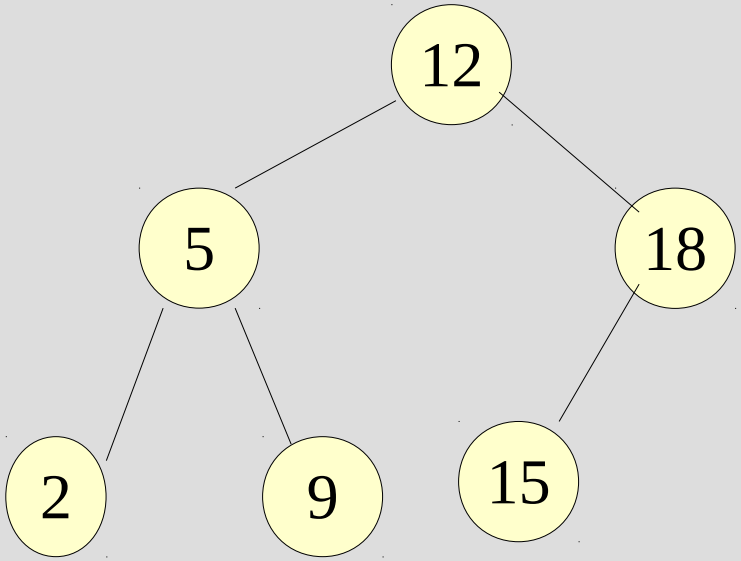
Cancellazione: funzionamento

- **3 casi possibili** (in ordine di complessità crescente):
  - Il nodo z è una foglia
  - Il nodo z ha un solo figlio
  - Il nodo z ha due figli

Nodo foglia

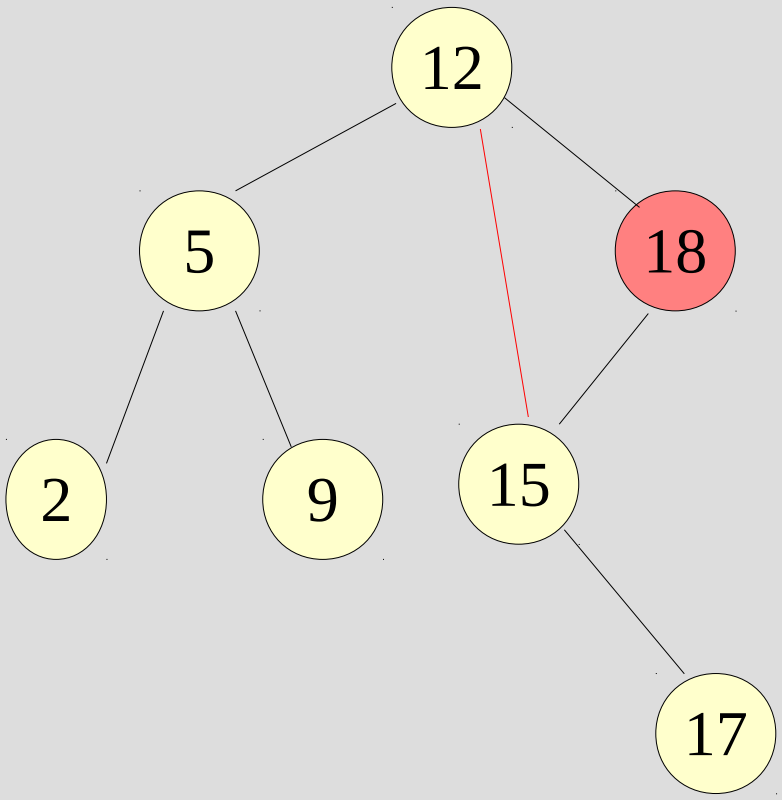
- Il nodo z viene semplicemente rimosso

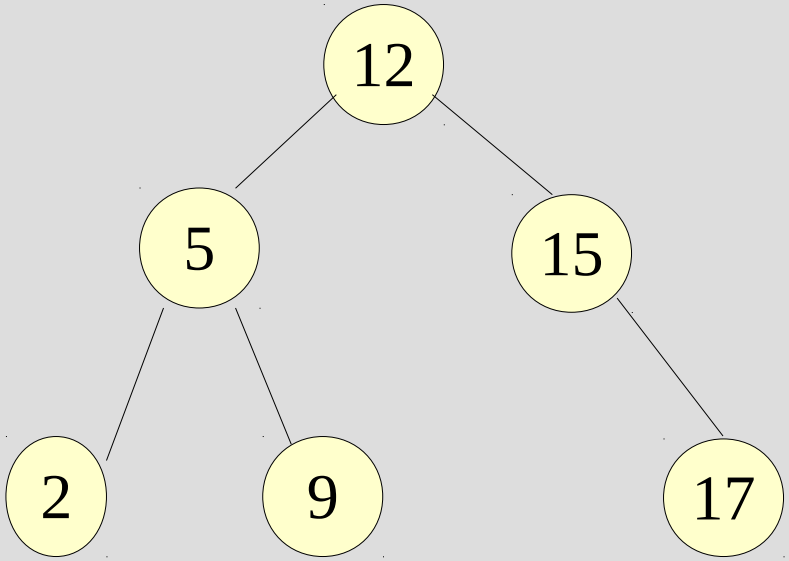




Nodo con un solo figlio

- Il nodo z viene cancellato creando un collegamento tra il padre e il figlio di z



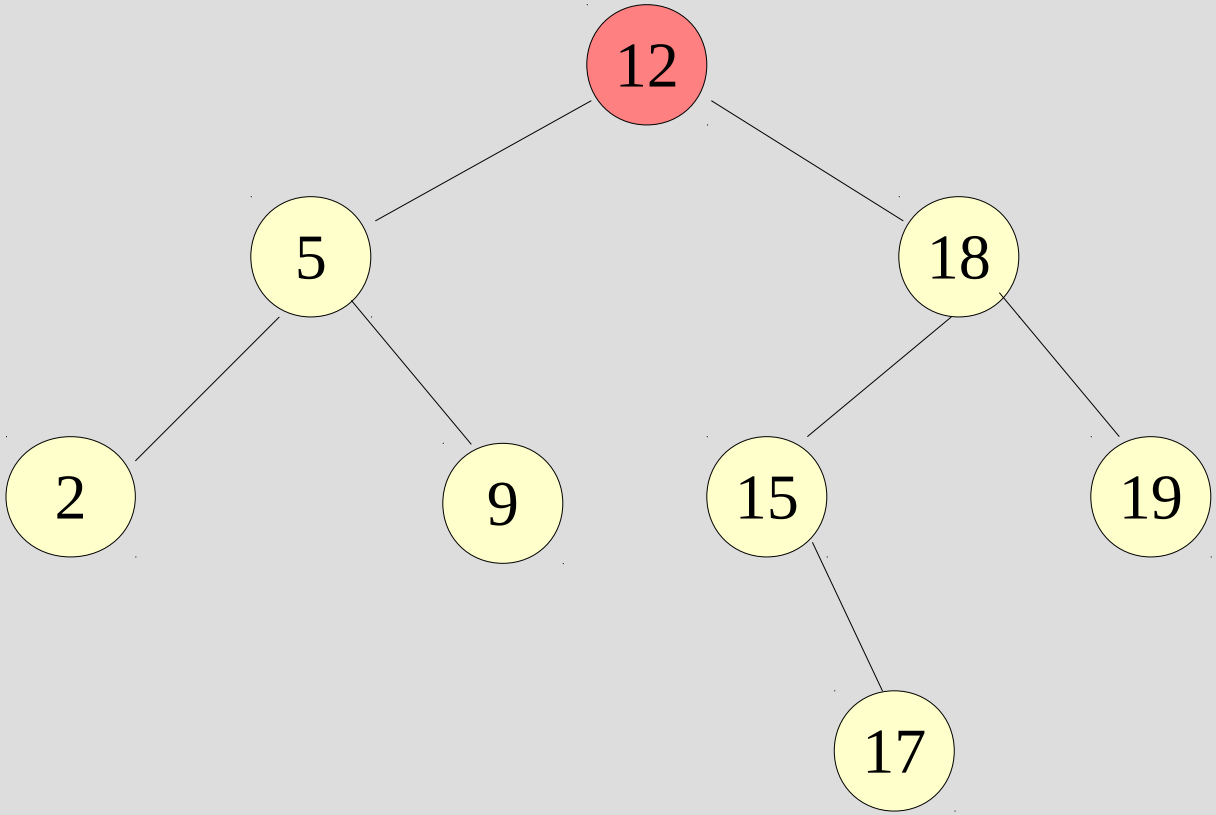


Nodo con due figli

- Cerchiamo il ***più piccolo dei suoi successori*** (o il ***più grande dei suoi predecessori***) per **sostituirlo**
- Il più piccolo dei successori di **z** è il nodo più a sinistra del sottoalbero destro di **z**
- Sicuramente il più piccolo dei successori non ha un figlio sinistro (***ha al più un figlio destro***)

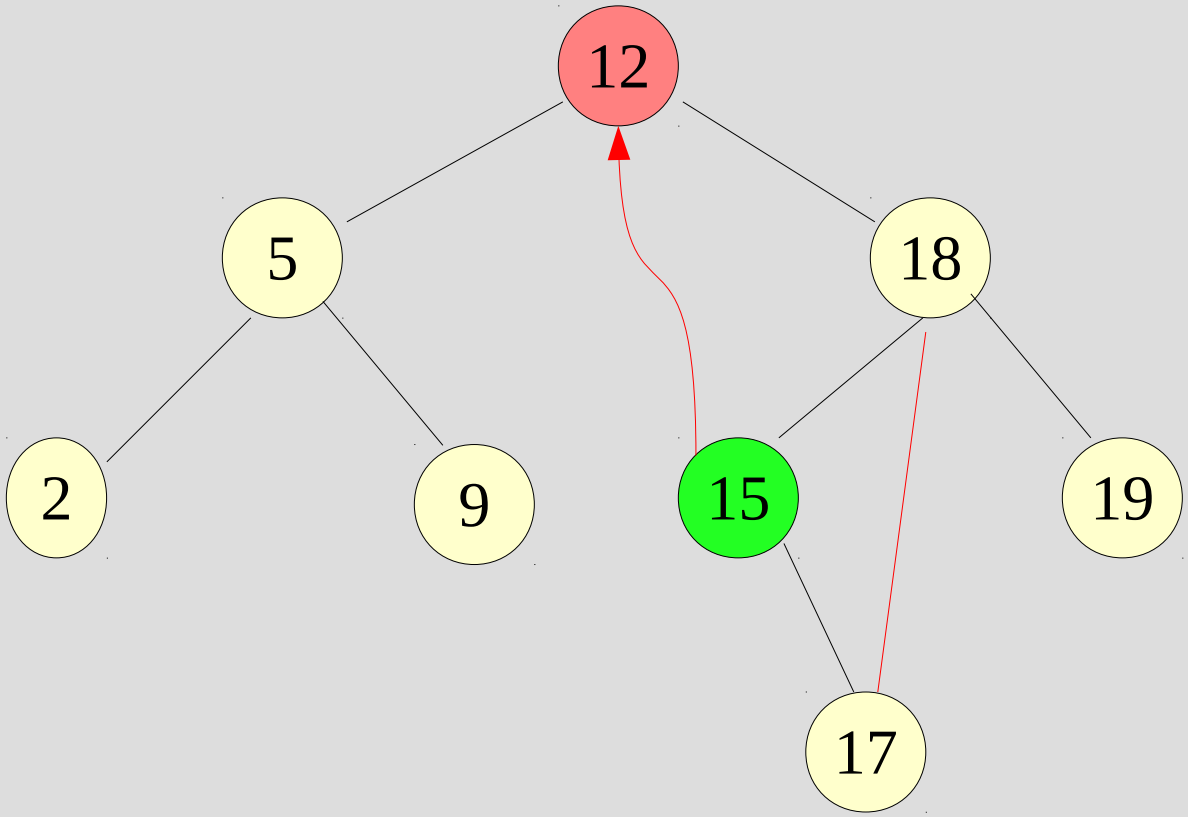
Esempio: nodo con due figli

- Eliminazione del valore 12



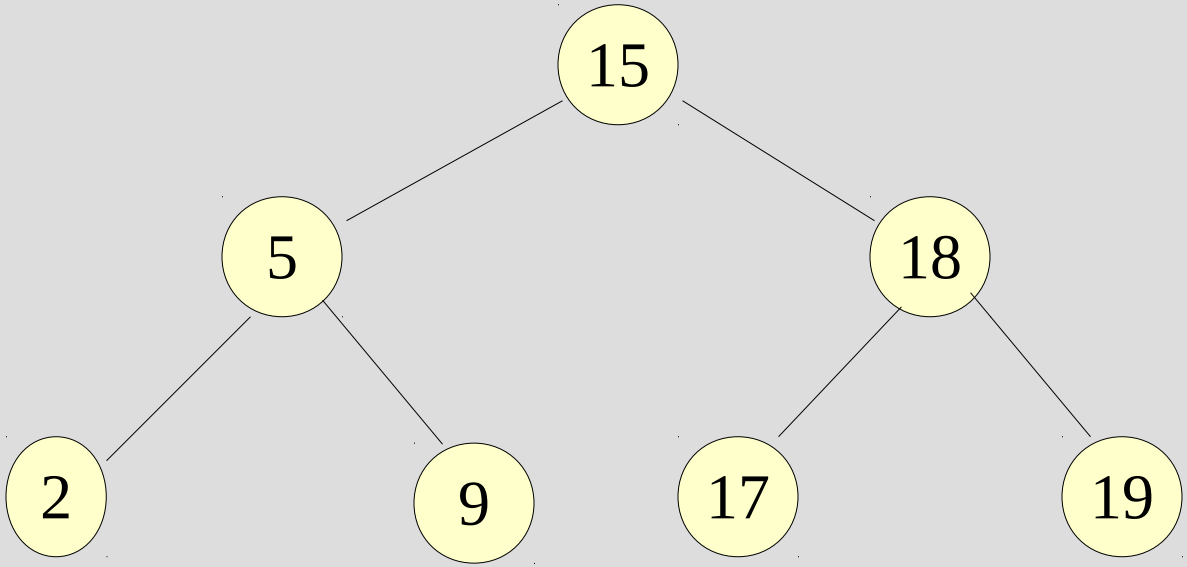
Esempio: nodo con due figli

- 15 è il più piccolo dei successori di z



Esempio: nodo con due figli

- Situazione finale



Programma

- *btree_compl.cc*
- Programma che permette di cancellare un elemento dell'albero avente un determinato valore di chiave
- Il programma mette inoltre a disposizione una funzione per deallocare correttamente l'intera struttura dell'albero (*delete_tree*)