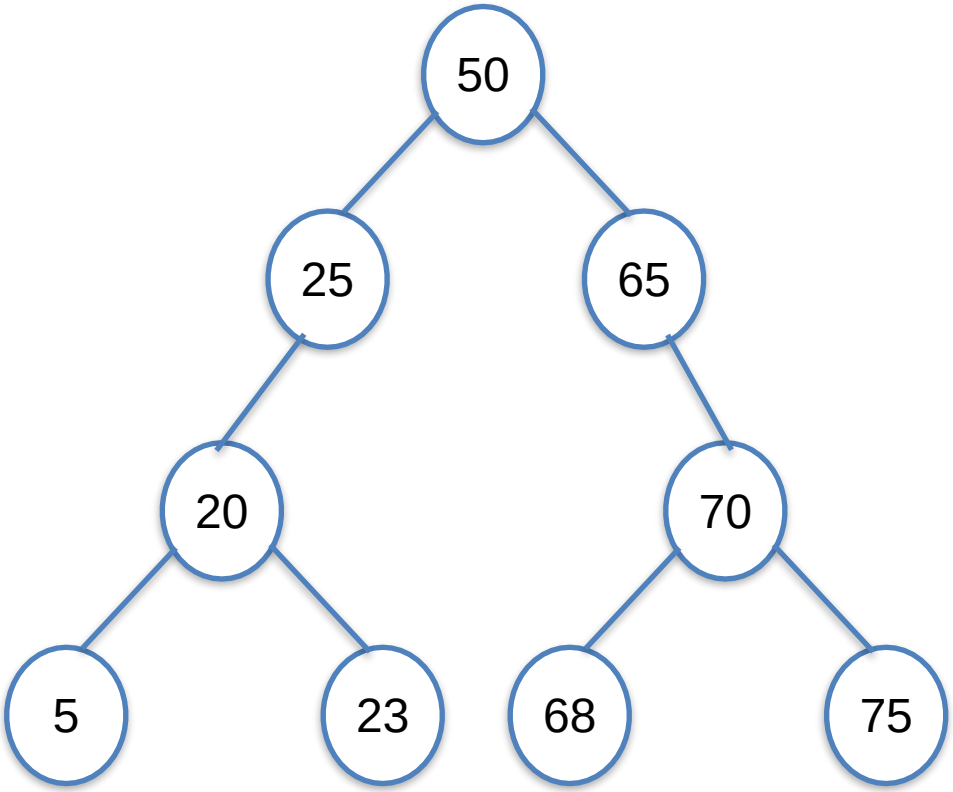
Binary Search Trees

BST property

- Keys in a BST satisfy the ***binary-search-tree property***

- Let *x* be a node in a binary search tree.
- If *y* is a node in the left subtree of *x* , then
  $y.key <= x.key$

- If y is a node in the right subtree of *x* , then
  $y.key >= x.key$



Querying a binary search tree

- **Query operations** - Search, Minimum, Maximum, Successor and Predecessor

- BST support these operations each one in time **O(h)** on any binary search tree of **height h.**

BST Insertion

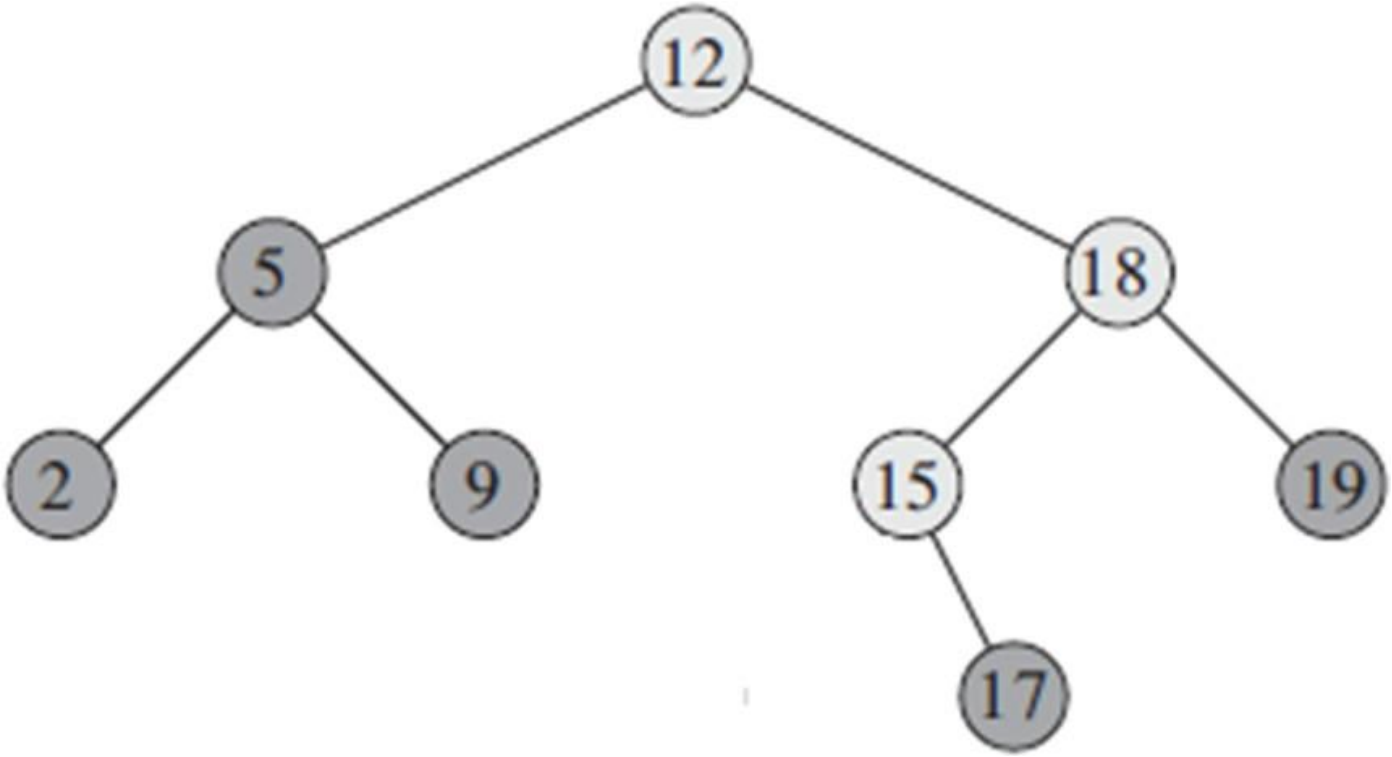
BST Insertion

- Insertion & Deletion are modifying operations
  - BST changes as the result of these operations
  - BST property continues to hold

Procedure to insert a value *v* into a BST T

- **Input**
  - BST *T*
  - A node z for which z.key = *v*, z.left = NIL & z.right =NIL
- **Tree-Insert** modifies
  - T
  - Some attributes of z
  - Inserts z into an appropriate position in T

Insert the node with key 13 to given T

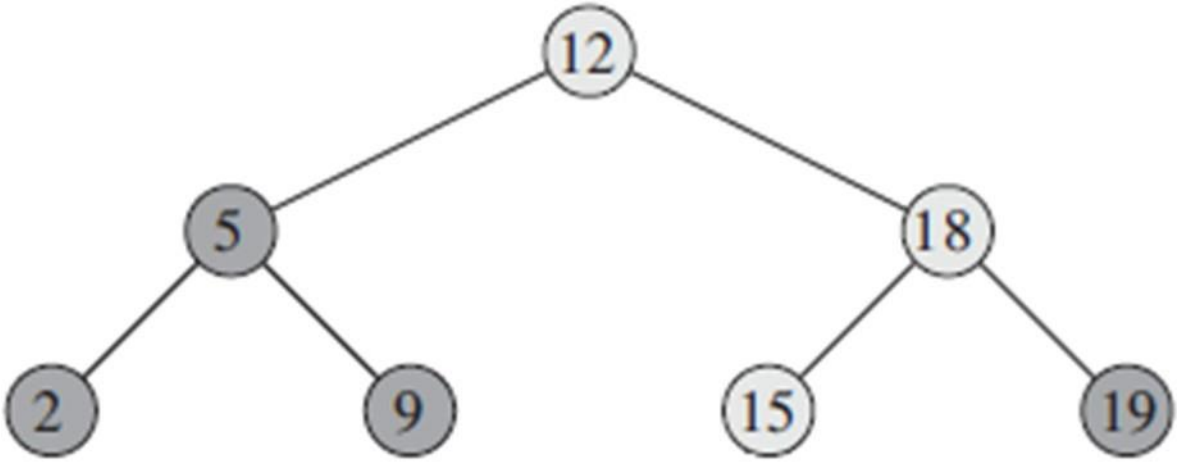


TREE-INSERT(T,z)

- Starts with the root of T
- Maintains two pointers x and y
- x traces a simple path downward looking for a NIL to replace with z
- y is the trailing pointer which is the parent of x
- Why do we need a trailing pointer y as the parent of x?

Another T

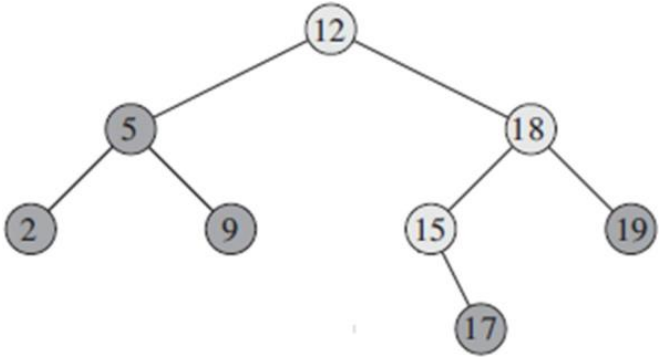
- Insert 3,17, 25, 8 in sequence (one after another)



TREE-INSERT(T,z)

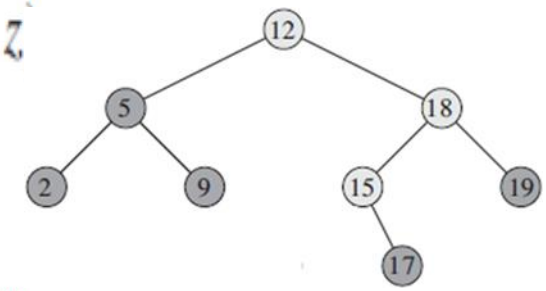
- Starts with the root of T
- Maintains two pointers x and y
- x traces a simple
- path downward looking for a NIL to replace with z
- y is the trailing pointer which is the parent of x

```
TREE-INSERT(T, z)
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4      y = x
 5      if z.key < x.key
 6          x = x.left
 7      else x = x.right
 8  z.p = y
 9  if y == NIL
10      T.root = z          // tree T was empty
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
```



TREE-INSERT(T,z) – contd.

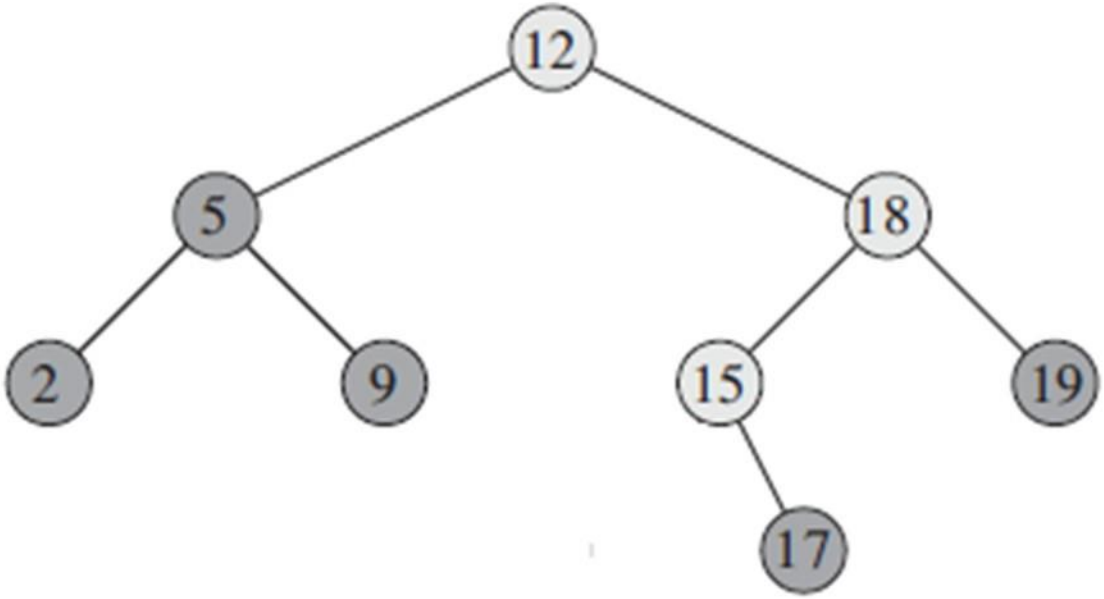
- While loop (lines 3-7) causes x & y to move down T until x becomes NIL
- NIL occupies the position where we need to insert z
- Lines 8-13 sets the pointers that cause z to be inserted




```
TREE-INSERT(T, z
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4      y = x
 5      if z.key < x.key
 6          x = x.left
 7      else x = x.right
 8  z.p = y
 9  if y == NIL
10      T.root = z          // tree T was empty
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
```

Running time of TREE-INSERT(T,z)

- Traverses from root to the appropriate position where z (eg: 13) has to be inserted



- Running time : O(h) on a tree of height h

Exercise

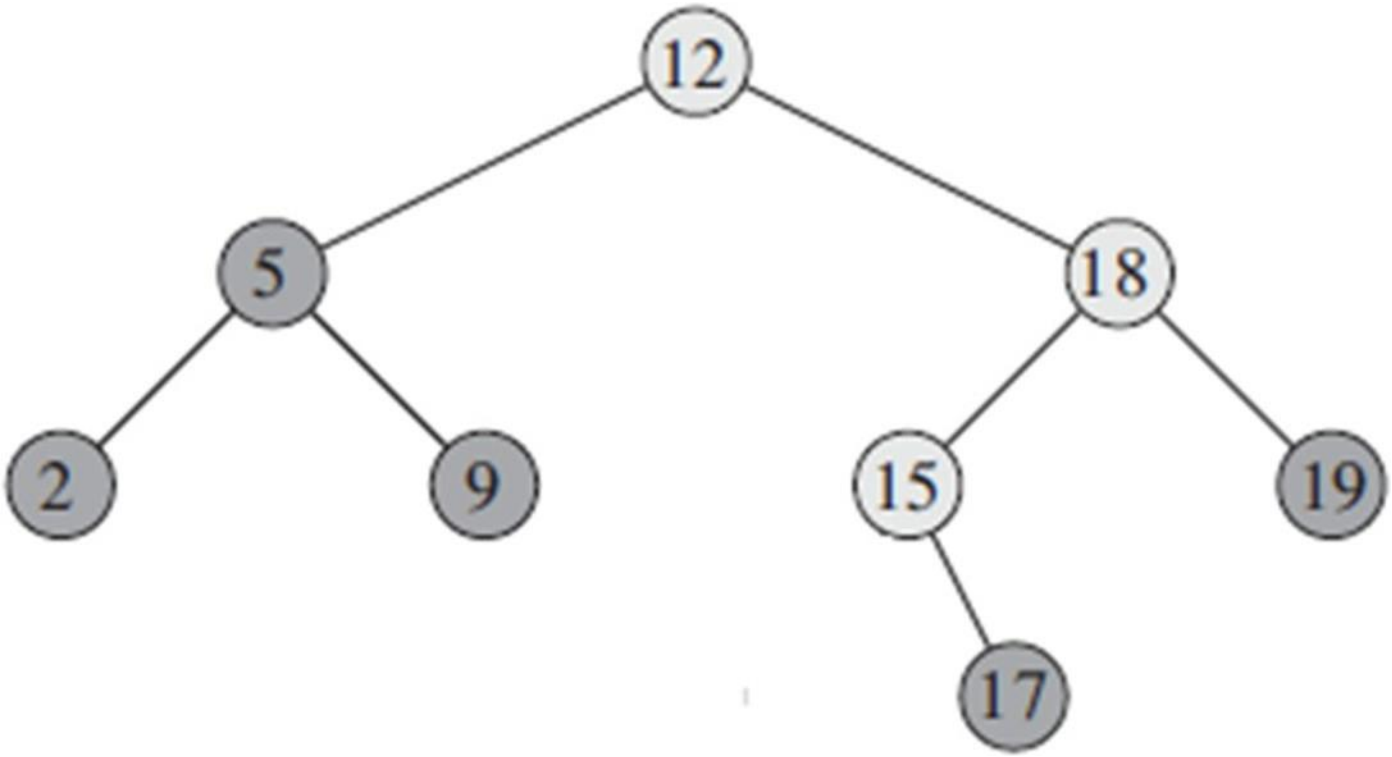
Write the recursive version of TREE-INSERT

BST Deletion

Overview

- **Binary Search Tree Deletion**
  - **Examples**
  - **Different cases**
  - **Algorithm**

Deletion of a node from a BST



References

- CLRS Book