Binary Search Trees

BST property

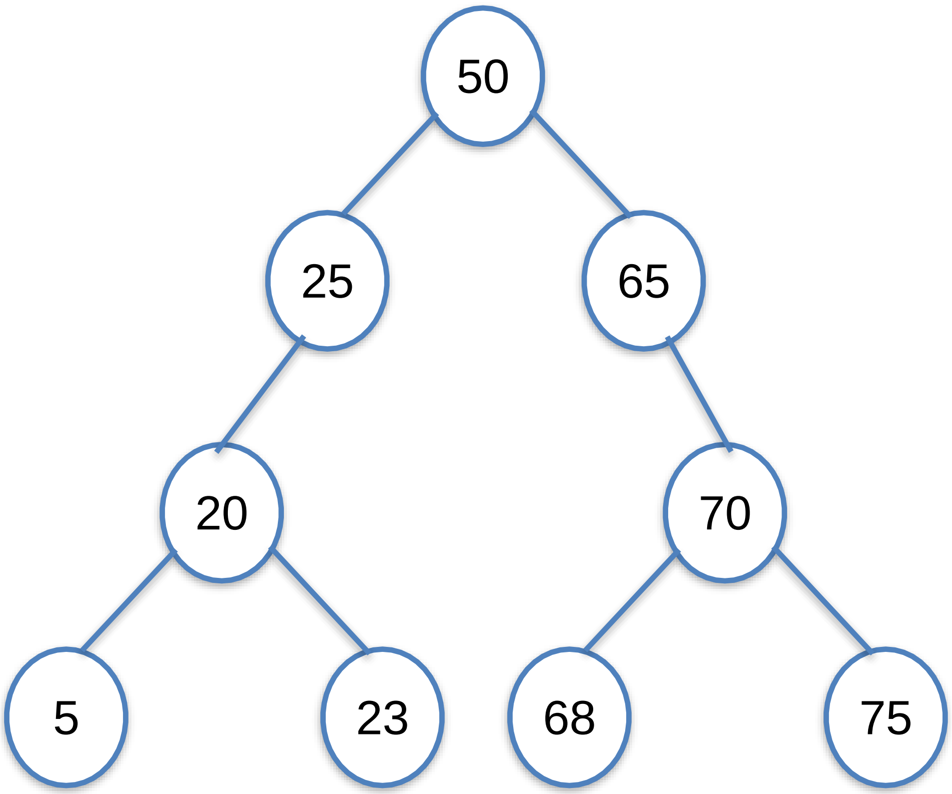
- Keys in a BST satisfy the ***binary-search-tree property***

- 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 operations on a BST

- **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.**

Modifying operations on BST

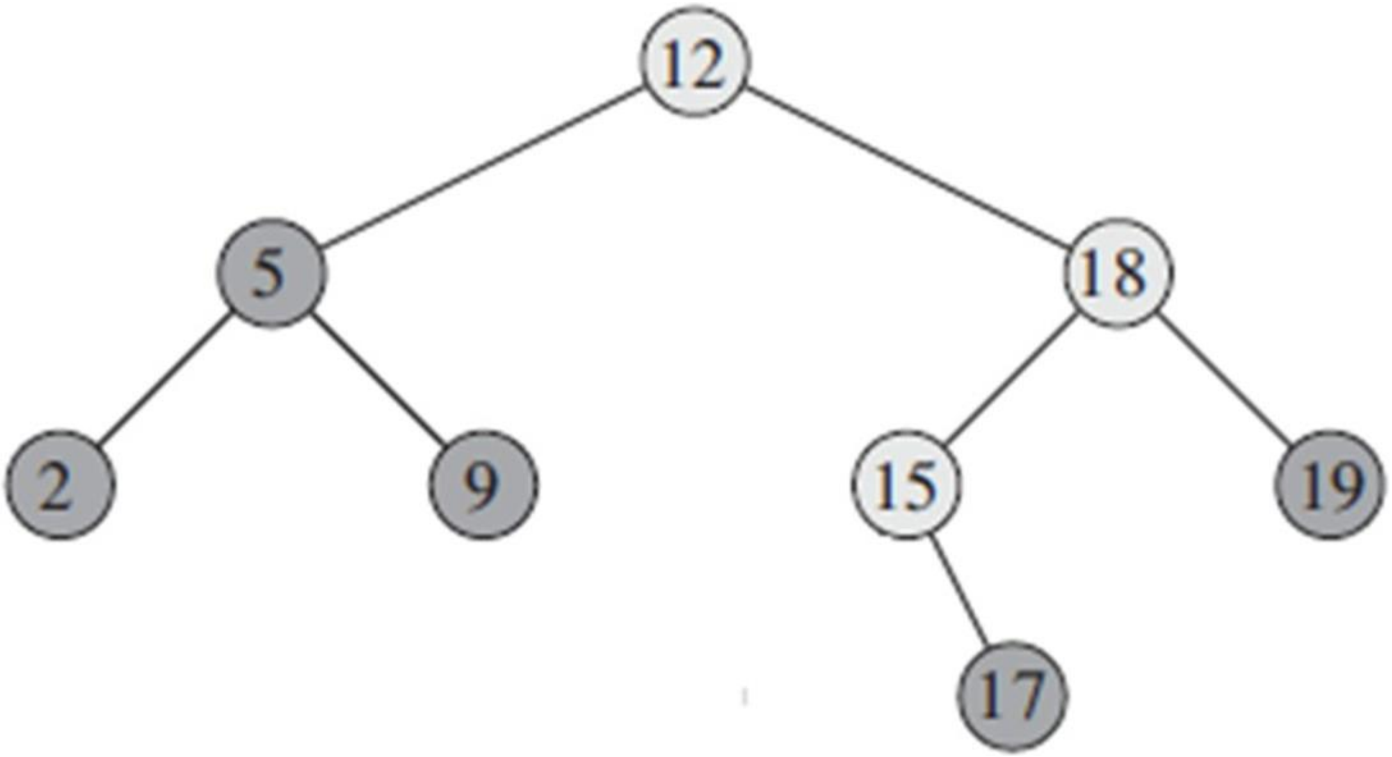
- Insertion – we have already discussed
- Deletion

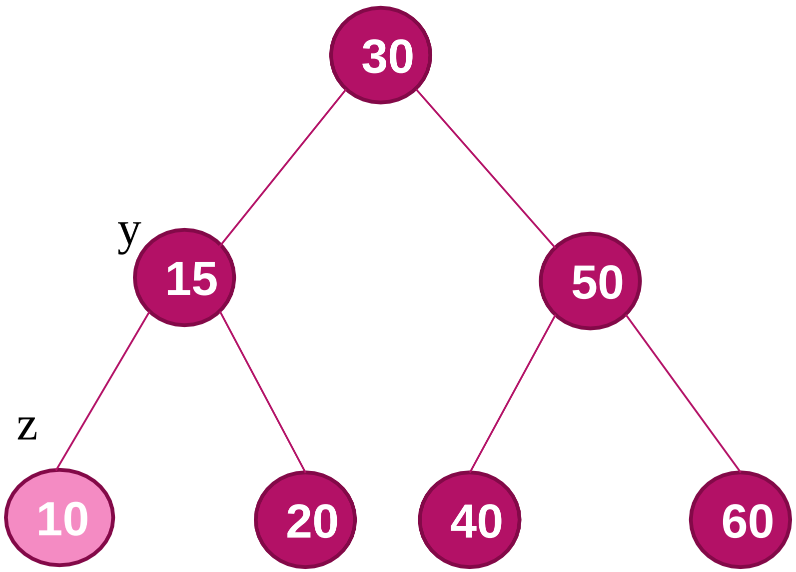
BST Deletion

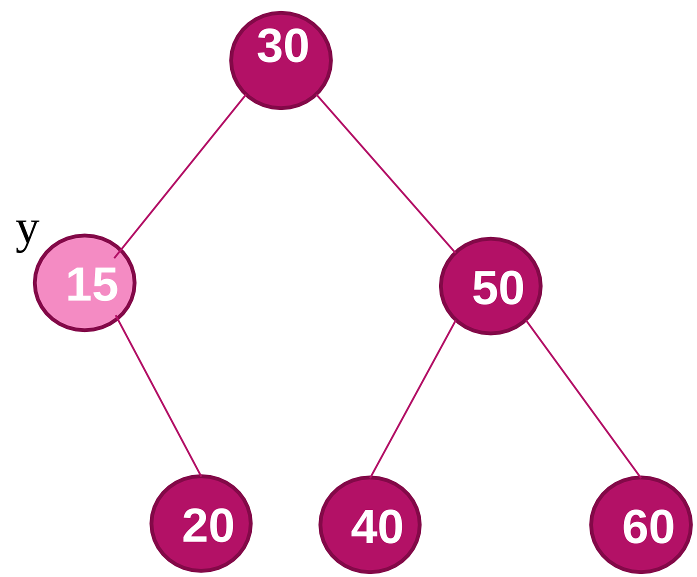
Overview

- Deletion of a node from a BST
  - **Examples**
  - **Different cases**
  - **Algorithm**

Deletion of a node from a BST



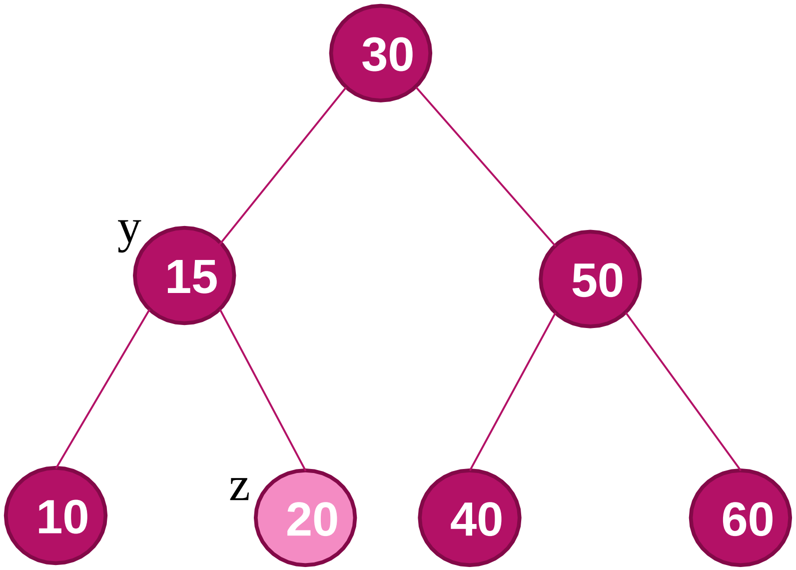


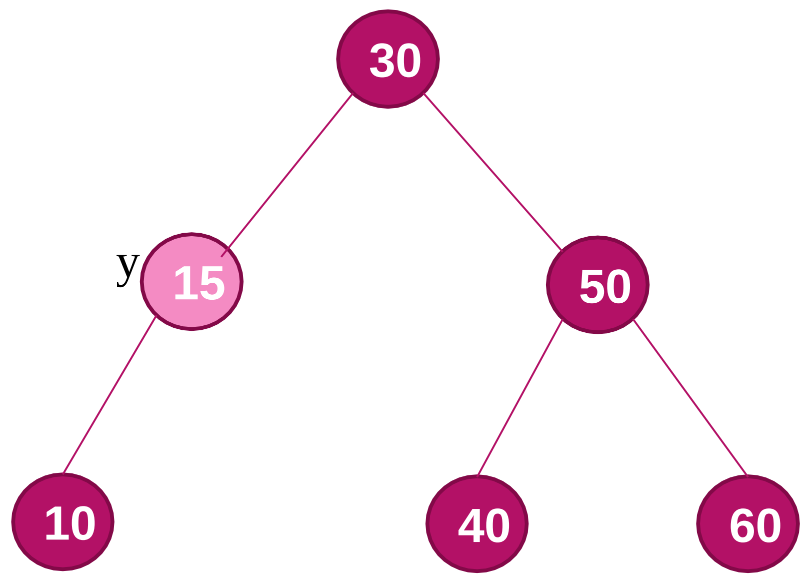


Delete z: a leaf node

Link updations ?

y.lchild to be set to **NIL**



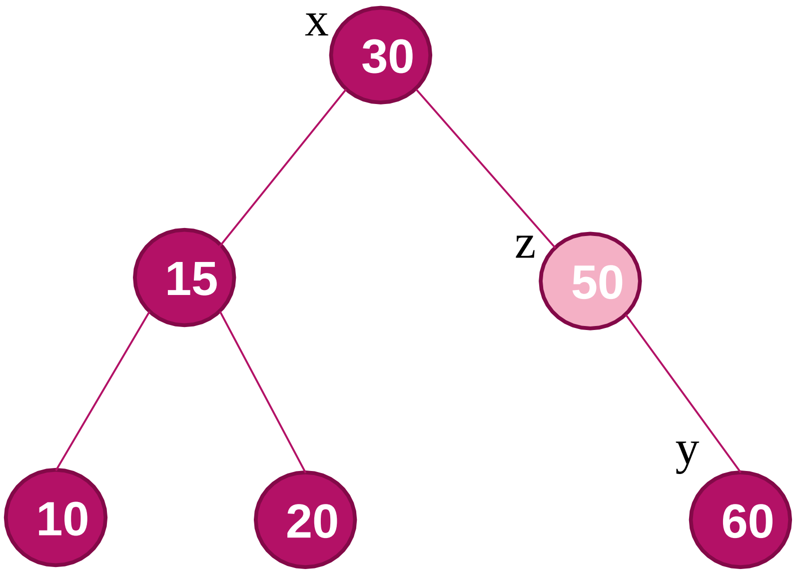


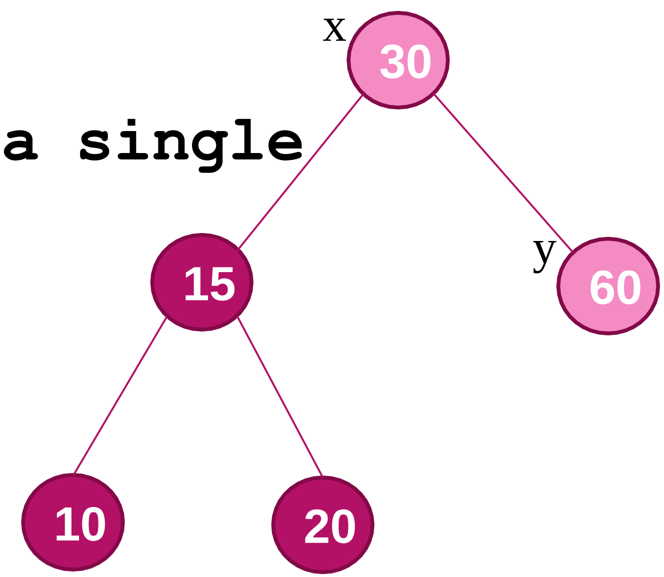
`delete z, a leaf node`

`y.rchild` to be set to **`NIL`**

BST Deletion - cases

- Deletion of
  - a leaf node
  - ....??

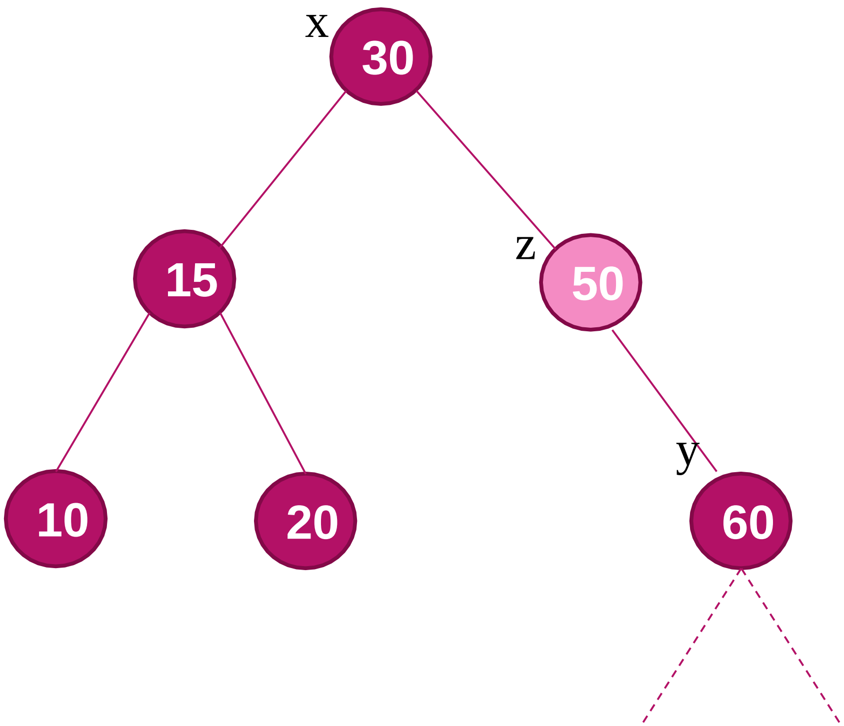




z: internal node with a single child

The lone child y can replace z

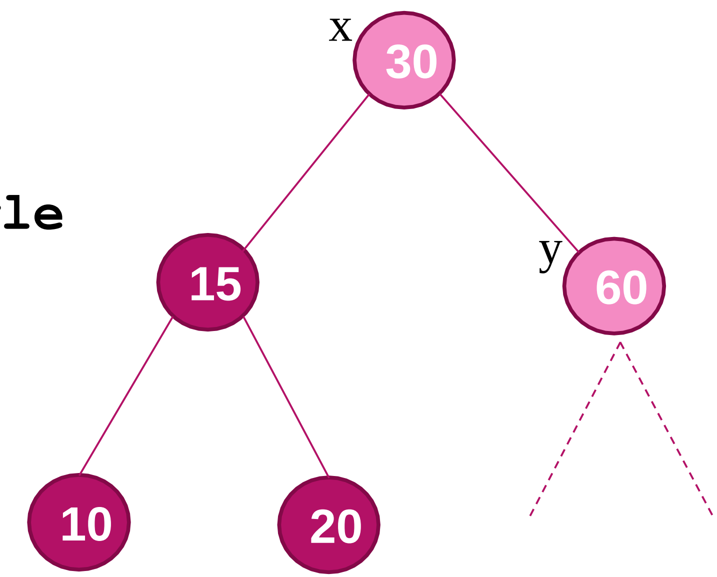
y.parent to be set as **x**
x.rchild to be set as **y**



`z: internal node with a single nonempty subtree`

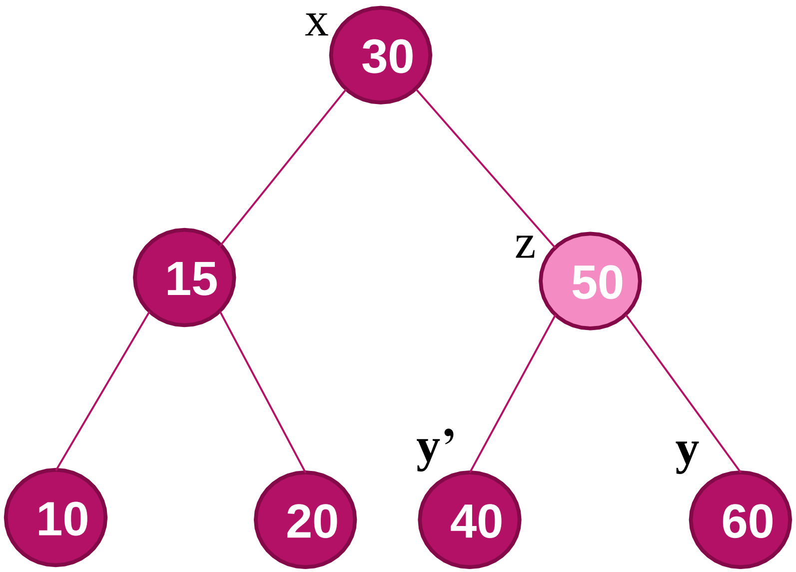
`y can replace z`

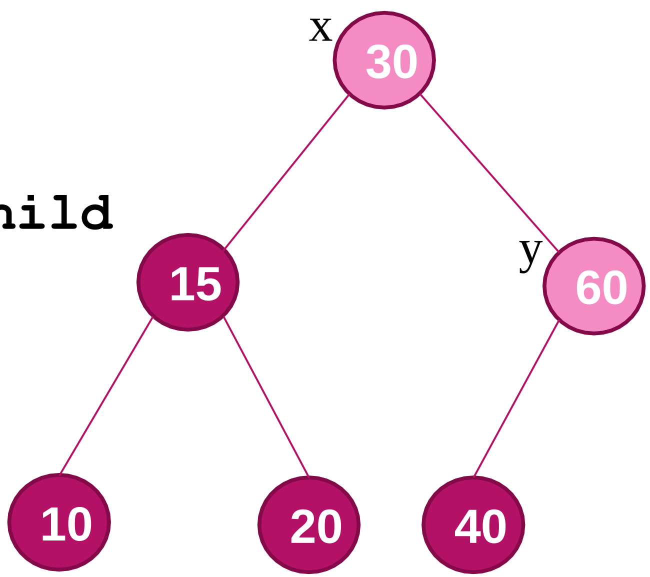
`y.parent` to be set as **`x`**
`x.rchild` to be set as **`y`**



BST Deletion - examples

- **Deletion of**
  - **a leaf node**
  - **a node with one child or one subtree**
  - ……………??





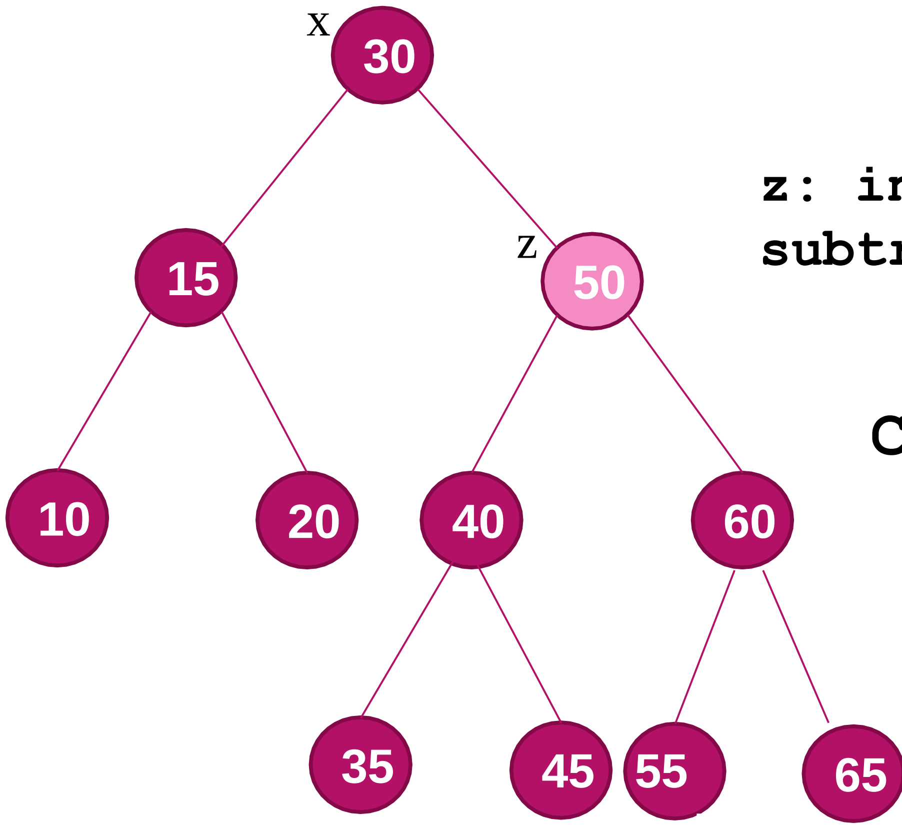
z:internal node with both left child and right child nonempty

y.parent to be set as x
x.rchild to be set as y

Alternatively can y′ replace z ?

BST Deletion - cases

- Deletion of
  - a leaf node
  - a node with one nonempty subtree(with a single child is included in this case)
  - a node with both the children

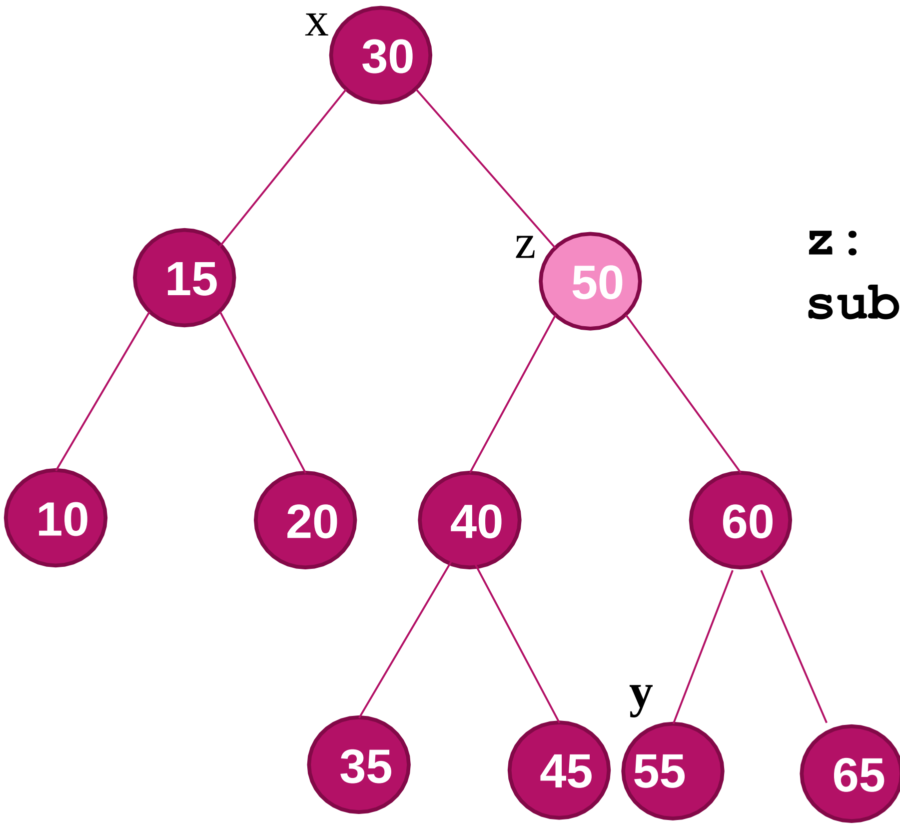


z: internal node, both the subtrees nonempty

Can not simply remove z

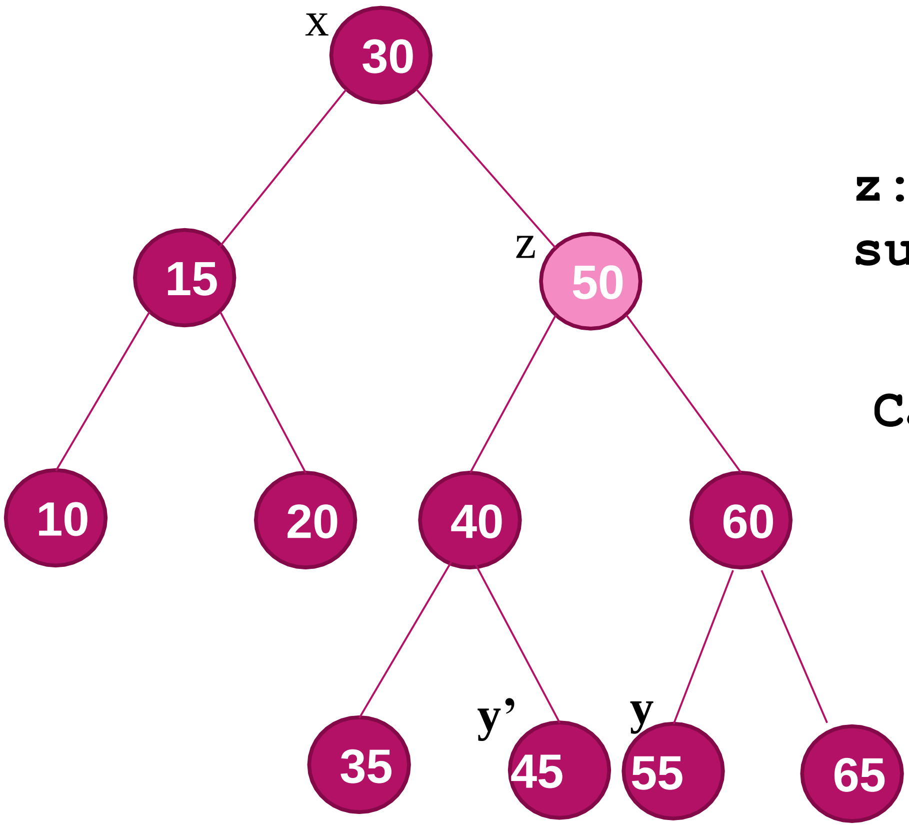
Can z be replaced by some other node y ?

Can we select a y from the subtree of z?



z: internal node, both the subtrees nonempty

Can z be replaced by y ?

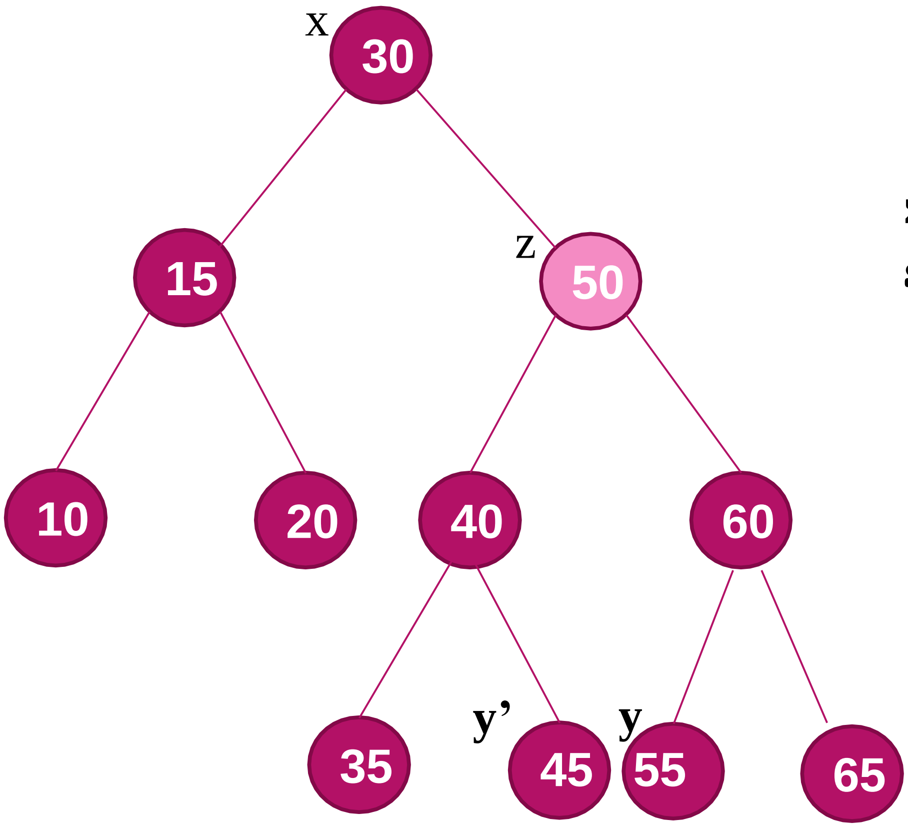


z: internal node, both the subtrees nonempty

Can z be replaced by y′?

Replacing z with z′s inorder successor/inorder predecessor

BST property is to be maintained



z: internal node, both the subtrees nonempty

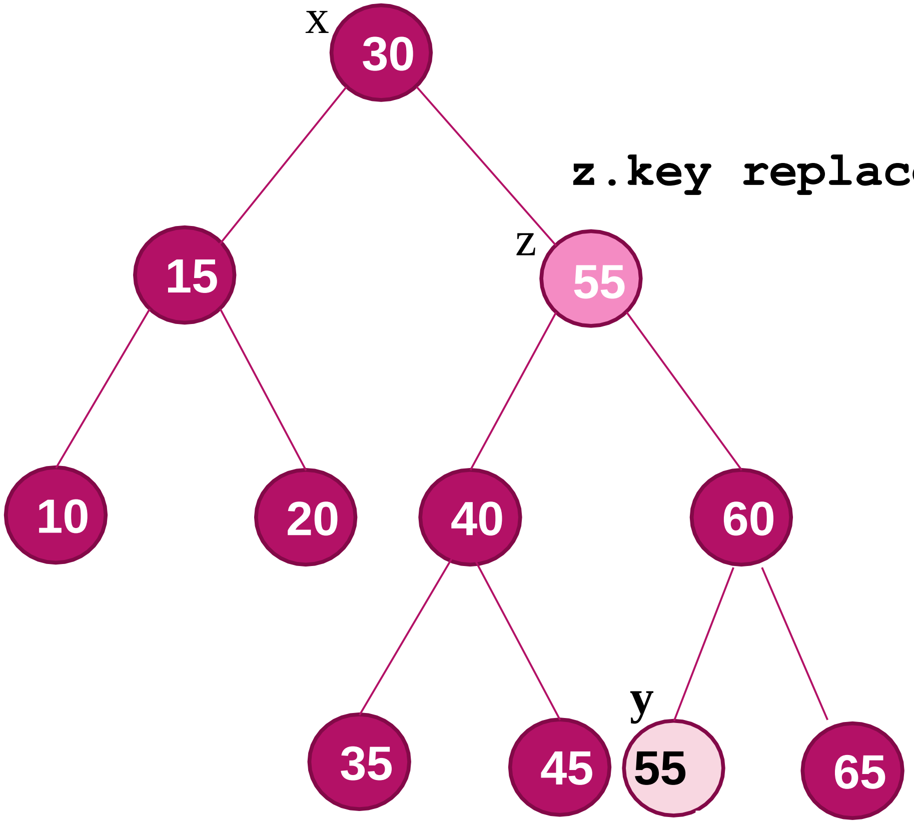
Solution1:

- Let y be z's inorder successor
- Replace z.key with y.key
- Delete y

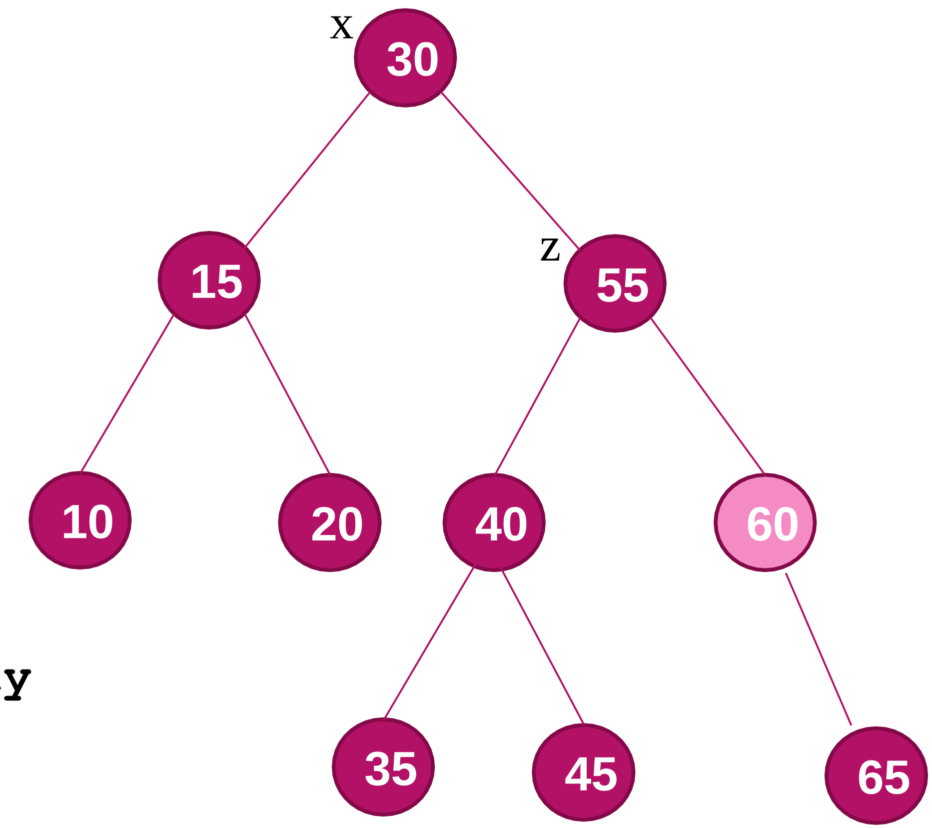
z is not physically removed

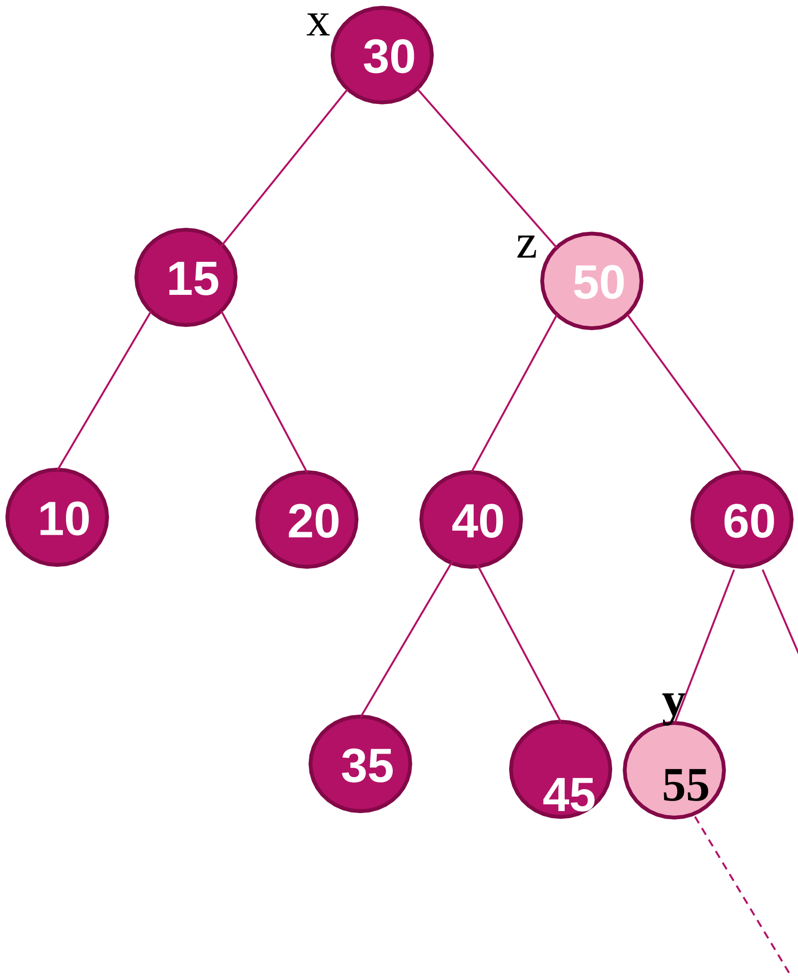
Is it correct to replace z.key with y'.key, where y' is the inorder predecessor of z and then remove y'?

z.key replaced with y.key



y physically
removed

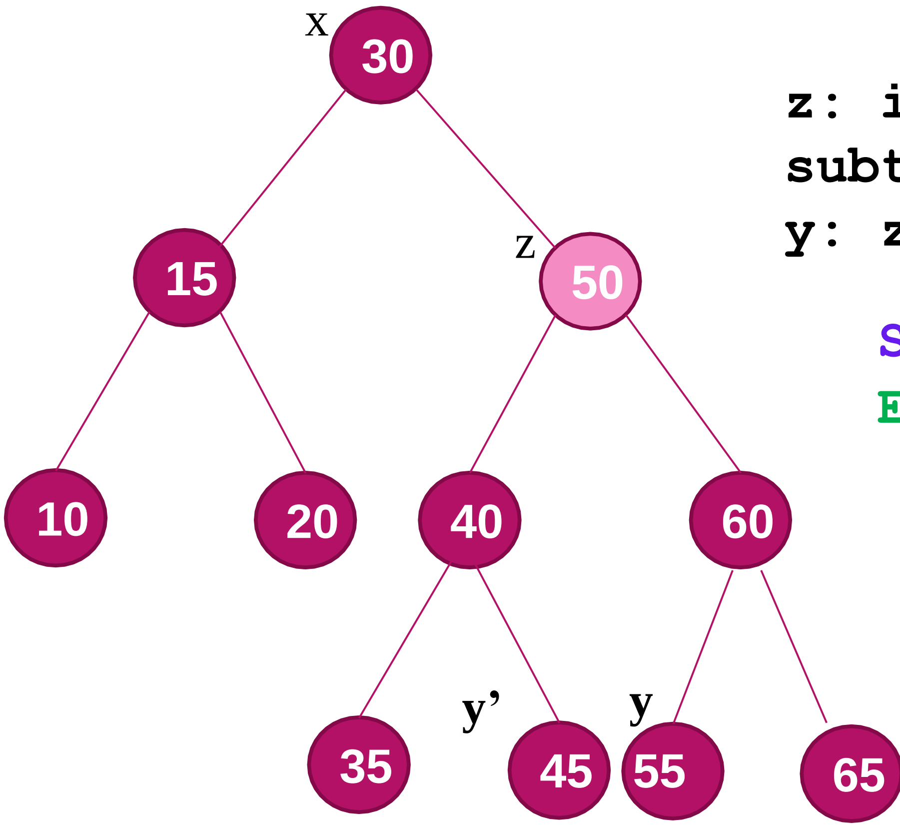




z: internal node with both the subtrees nonempty

z's successor has no left child, can have a nonempty right subtree

Physically deleted node has at most one nonempty subtree



z: internal node, both the subtrees nonempty.
y: z's inorder successor

Solution #1 (CLRS 2nd edn.):
Exercise for you

- Copy data in y to z
- Delete y
- z is not physically removed

Solution #2 (CLRS 3rd edn.):
- Node z replaced by node y
- Node y is not deleted

BST Deletion - Algorithm :Cases

- **Let z be the node to be deleted**

   a. z has no left child

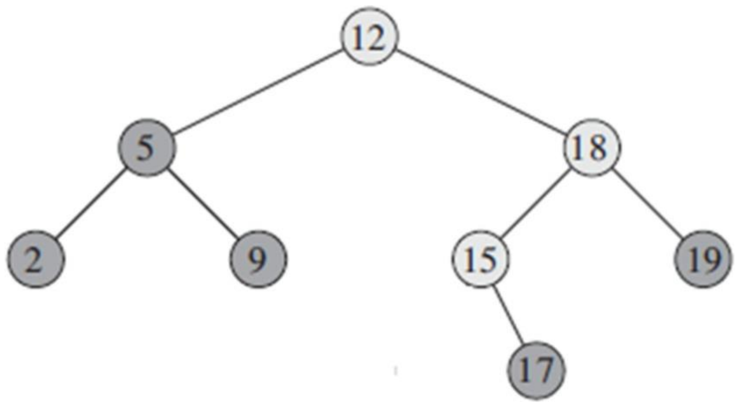
   b. z has just one child, which is its left child

   c. z has both a left and a right child

z being a leaf node - taken care of in case a(right child can be empty on non empty)

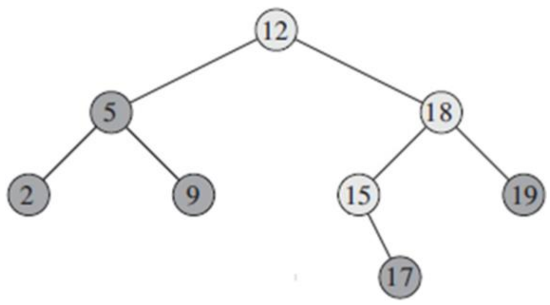
BST Deletion :Cases a and b

```
TREE-DELETE(T, z)

    if z.left == NIL     //right child may or may not be empty
        TRANSPLANT (T, z, z.right)

    elseif z.right == NIL   //z has left child, no right child

        TRANSPLANT (T, z, z.left)

    else ……… //z has both children, split into two cases c and d
```



BST Deletion :TRANSPLANT

```
TRANSPLANT(T, u, v) // replaces the subtree rooted at u with
                    // the subtree rooted at v

   if u.p == NIL     // u is root

       T.root = v

  elseif u == u.p.left  //u is lchild of its parent

      u.p.left = v

  else u.p.right = v

  if v ≠ NIL

      v.p = u.p     // v.left,  v.right updations, if required,
                    // to be done by the caller
```



RecallBST Deletion - Cases

- Let z be the node to be deleted

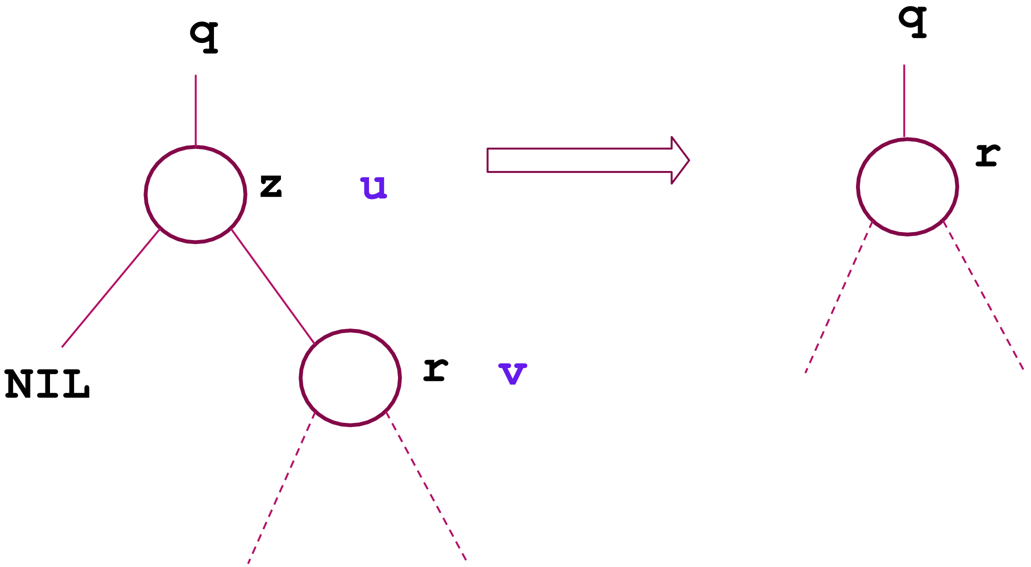
a. z has no left child

b. z has just one child, which is its left child

c. z has both a left and a right child

z being a leaf node - taken care of in case 1(right child can be empty on non empty)

(a)




```
if z.left == NIL
        TRANSPLANT (T, z, z.right)
```

TRANSPLANT(T, u, v)replaces the subtree rooted at u with the subtree rooted at v

Recall BST Deletion - Cases

- Let z be the node to be deleted

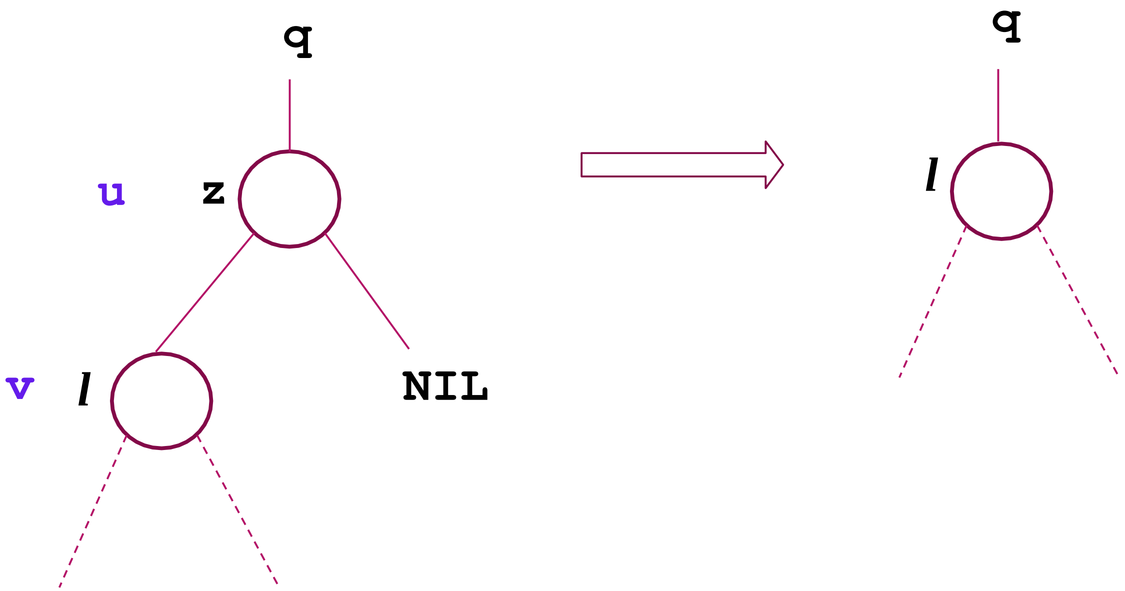
a. z has no left child

b. z has just one child, which is its left child

c. z has both a left and a right child

z being a leaf node – taken care of in case 1(right child can be empty on non empty)

(b)




```
…….elseif z.right == NIL
          TRANSPLANT (T, z, z.left) //z has left child l
```

Recall BST Deletion - Cases

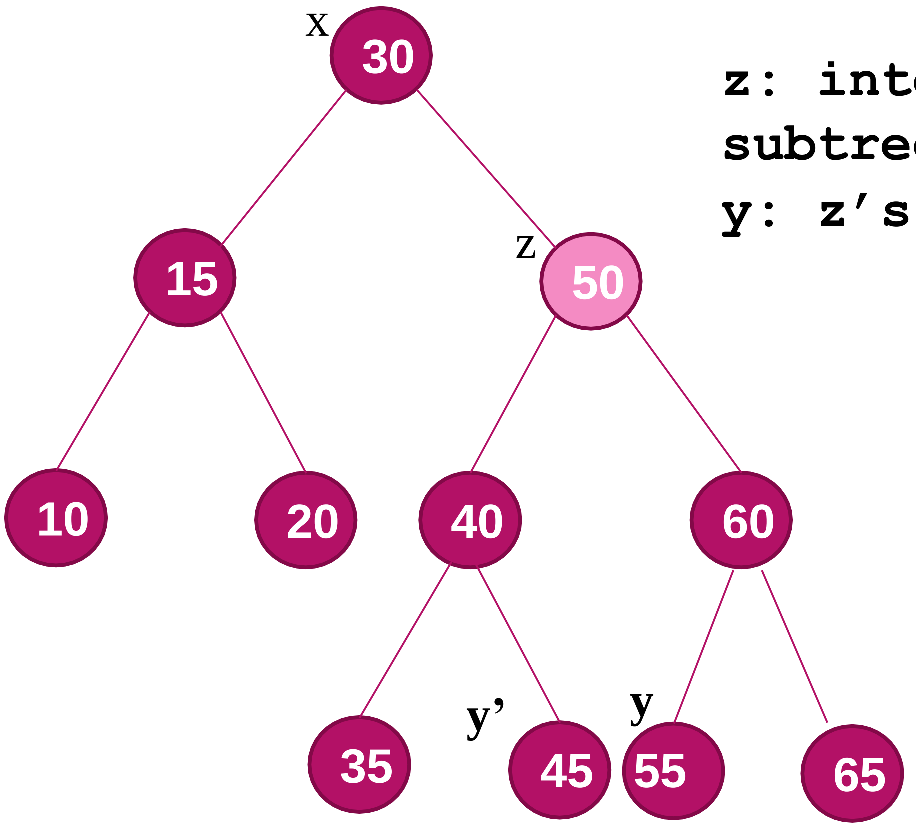
- Let z be the node to be deleted

a. z has no left child

b. z has just one child, which is its left child

c. z has both a left and a right child

z being a leaf node - taken care of in case 1(right child can be empty on non empty)



z: internal node, both the subtrees nonempty.
y: z's inorder successor

Solution #2 (CLRS 3rd edn.):

- Node z replaced by node y
- Node y is not deleted

BST Deletion: Solution #2

```
TREE-DELETE(T, z)
   if z.left == NIL     //right child may or may not be empty
       TRANSPLANT (T, z, z.right)
   elseif z.right == NIL  //z has left child, no right child
       TRANSPLANT (T, z, z.left)
   else .........
       //z has both children, find z's successor y, replace z by y
```

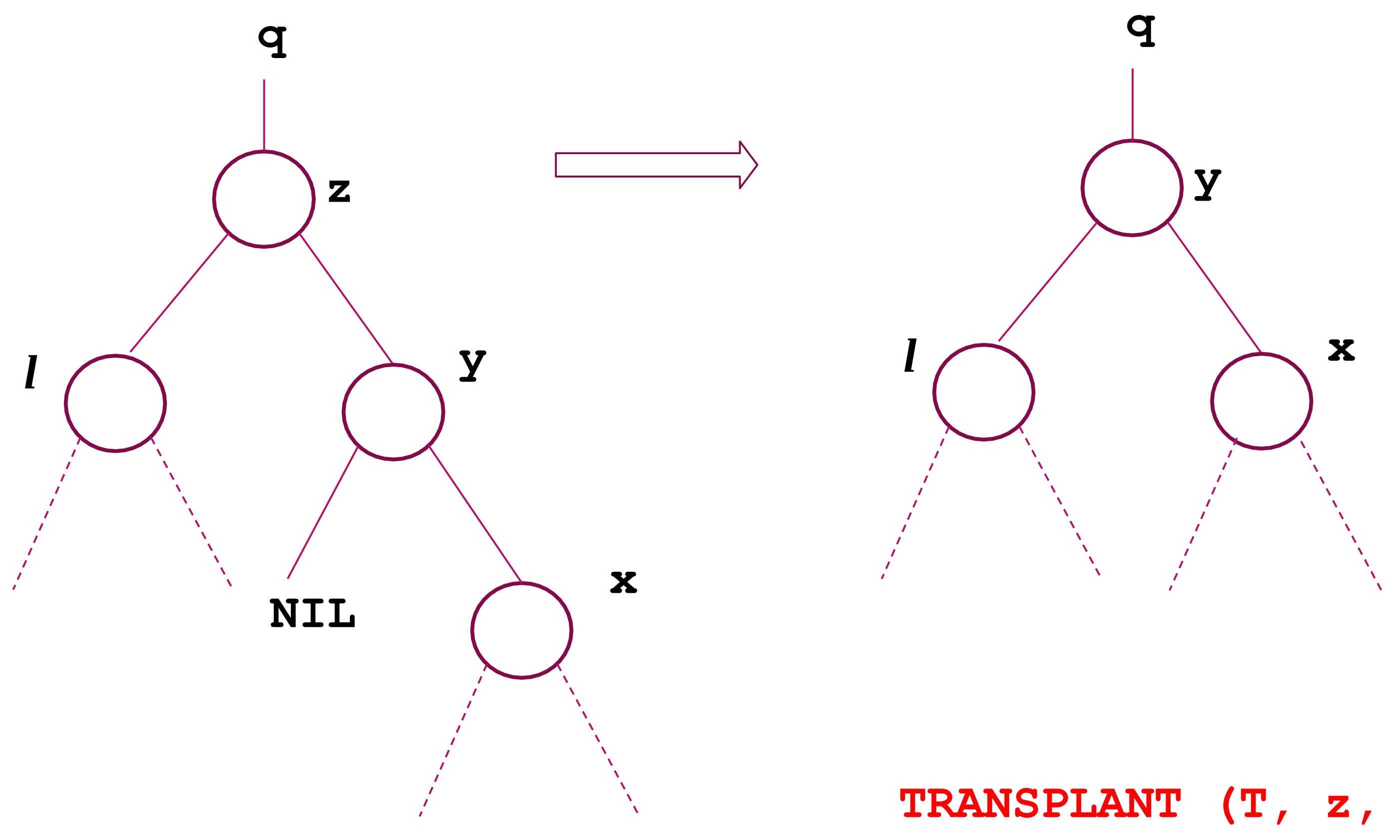
Split in to cases c andd

z has both a left and a right child : find z's successor y

(c) y is z's right child: replace z by y

(d)y is not right child of z, y lies within the right subtree of z: replace y by its own right child. Replace z by y

(c) Successor y is z's right child ie. if (y.p == z)



TRANSPLANT (T, z, y)
y.left = z.left
y.left.p = y

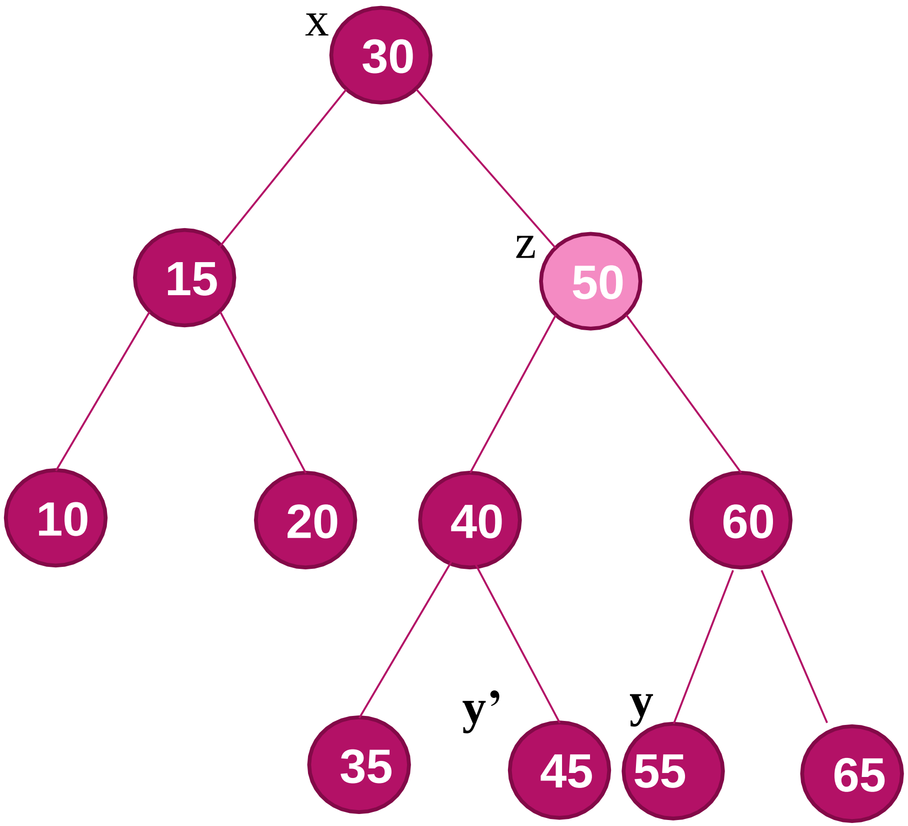
```
TREE-DELETE(T, z)
if z.left == NIL    ......
elseif z.right == NIL   ......
else y= TREE-MINIMUM(z.right) //z has both children, find z's successor y
      if (y.p == z)    // y is right child of z - case (c)
           TRANSPLANT (T, z, y)
           y.left = z.left
           y.left.p = y
     else ......
```

Split in to cases c andd

z has both a left and a right child : find z's successor y

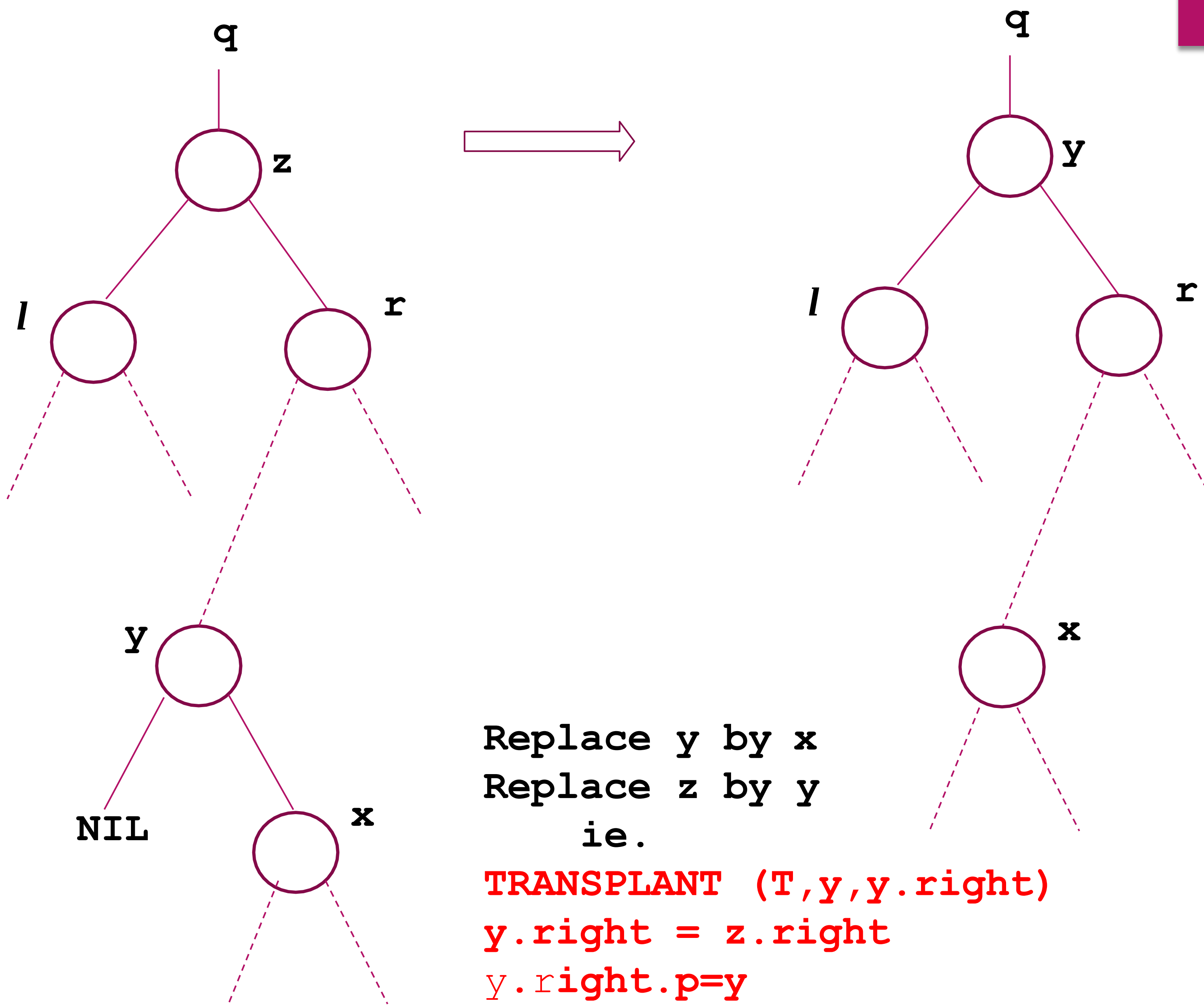
(c) y is z's right child: replace z by y

(d)y is not right child of z, y lies within the right subtree of z: replace y by its own right child. Replace z by y



Example of a BST depicting the case specified in previous slide

(d) Successor y is not right child of z, y lies in the subtree rooted at r(r is z′s right child)ie. if $(y.p \neq z)$



Replace y by x
Replace z by y
ie.
TRANSPLANT (T,y,y.right)
y.right = z.right
y.right.p=y

```
TREE-DELETE(T, z)

if z.left == NIL    ......

elseif z.right == NIL   ......

else y= TREE-MINIMUM(z.right) //z has both children, find z's
successor y

      if (y.p ≠ z)    // y is not right child of z

          TRANSPLANT (T, y, y.right)

          y.right = z.right

          y.right.p=y
```

TRANSPLANT (T, z, y)
y. left = z.left
y.left.p = y

```
TREE-DELETE(T, z)
if z.left == NIL   ......
elseif z.right == NIL  ......
else y= TREE-MINIMUM(z.right) //z has both children, find z's
                              // successor y
        if (y.p ≠ z) // y is not right child of z

          TRANSPLANT (T, y, y.right)
          y.right = z.right                          (d)
          y.right.p = y


      TRANSPLANT (T, z, y)
      y.left = z.left
      y.left.p = y             (c)
```

```
TREE-DELETE(T, z)
if z.left == NIL    ......
elseif z.right == NIL   ......
else y= TREE-MINIMUM(z.right) //z has both children, find z's
                              //successor y
      if (y.p ≠ z)    // y is not right child of z

         TRANSPLANT (T, y, y.right)
         y.right = z.right
         y.right.p = y
  TRANSPLANT (T, z, y)
  y.left = z.left
  y.left.p = y
```

These 4 link updates are not part of TRANSPLANT

Reference

1. **T H Cormen, C E Leiserson, R L Rivest, C Stein *Introduction to Algorithms*, 3rd ed., PHI, 2010**