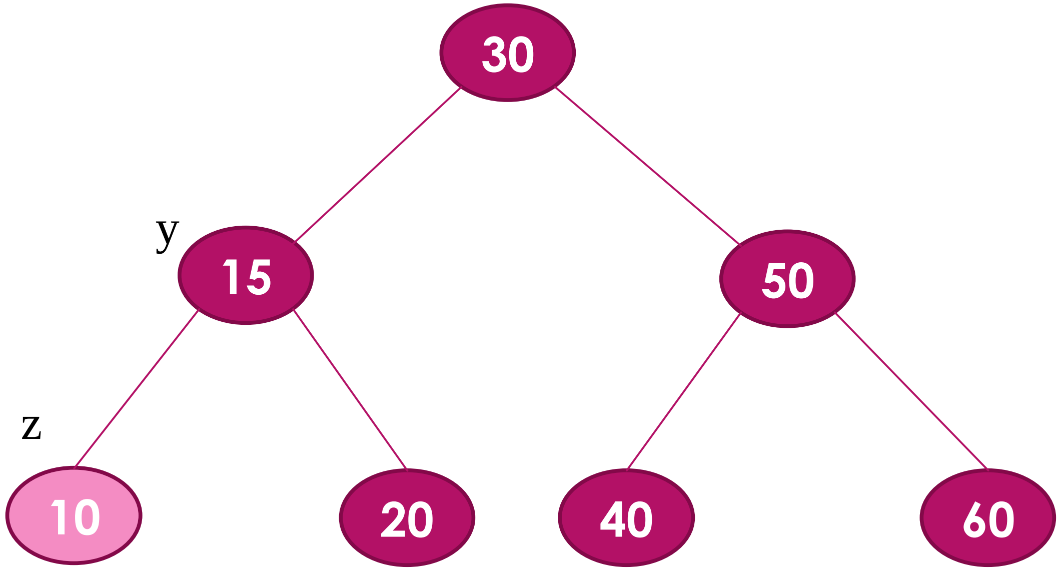
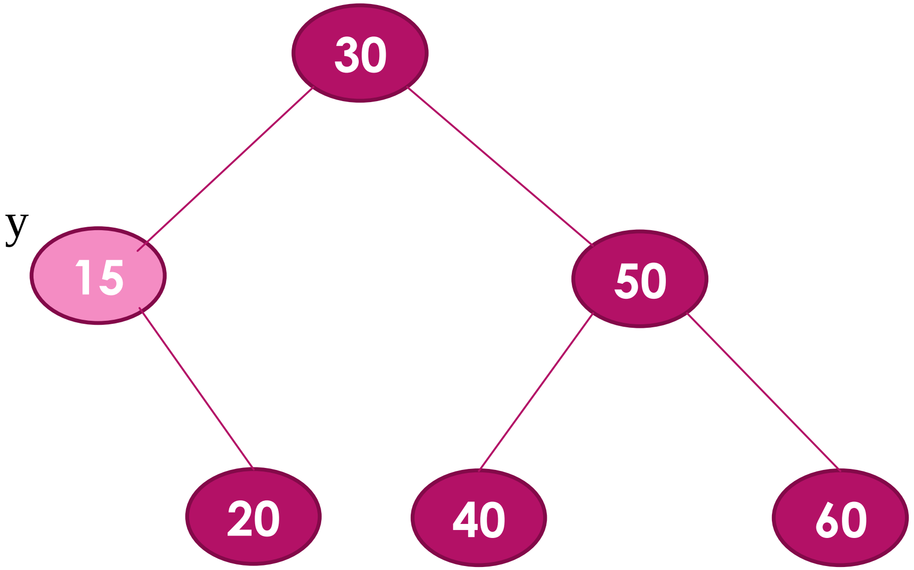
Overview

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

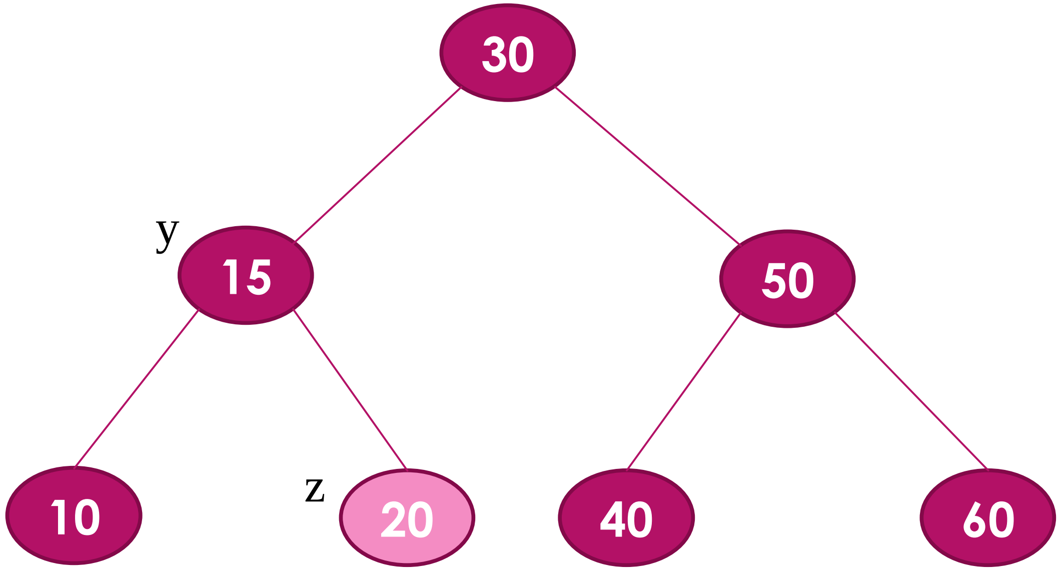




Delete z: a leaf node

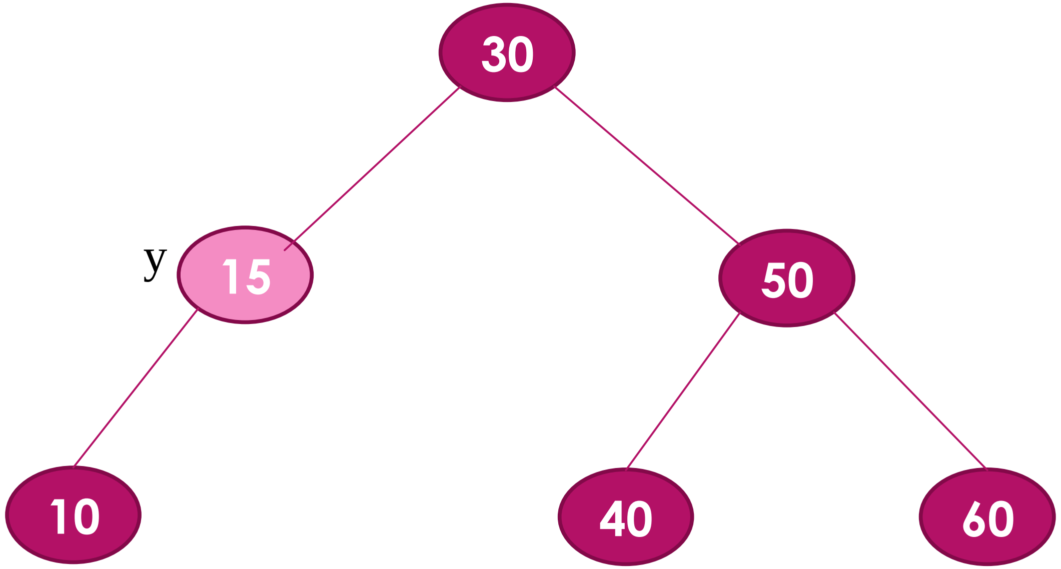
Link updations ?

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



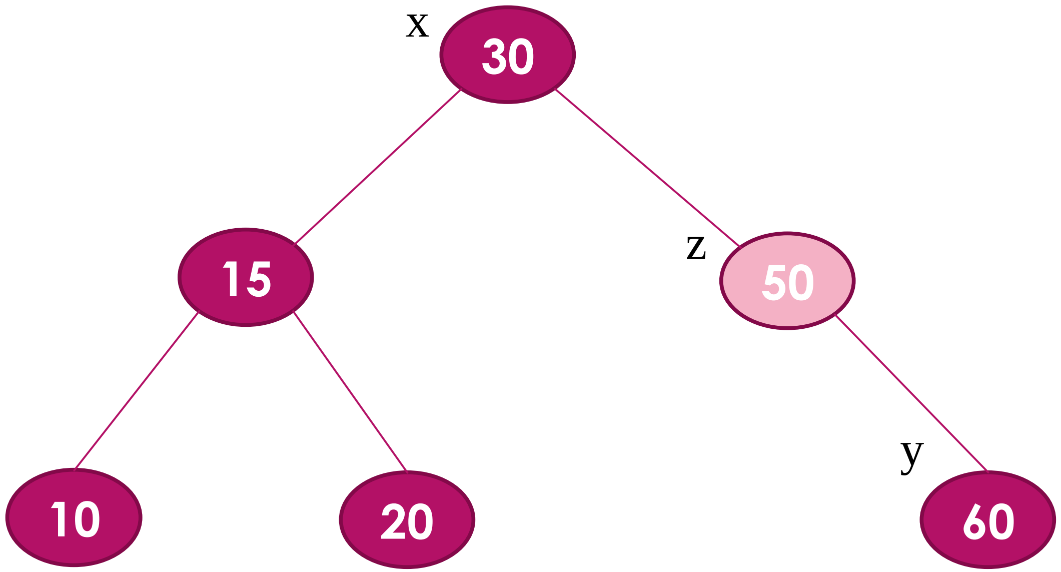
delete z, a leaf node

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



BST Deletion - examples

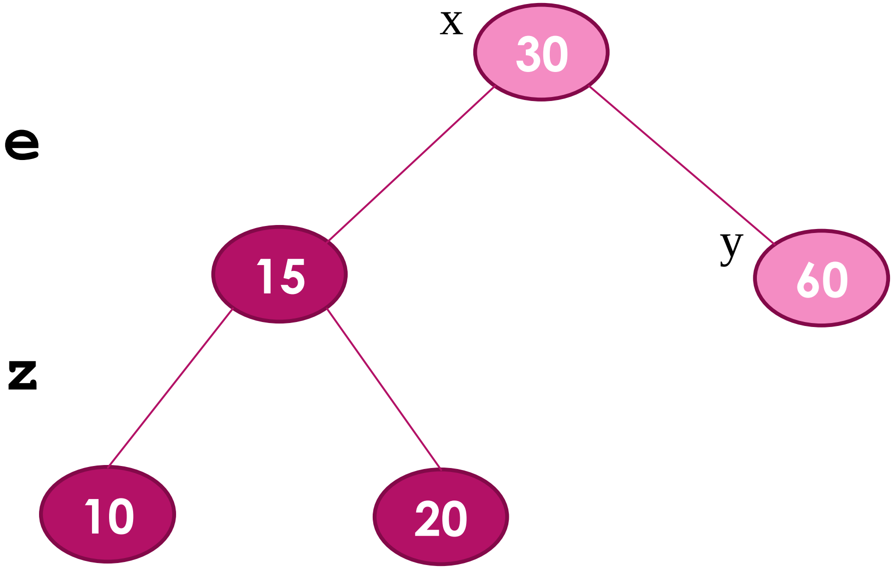
- Deletion of
  - a leaf node
  - …………

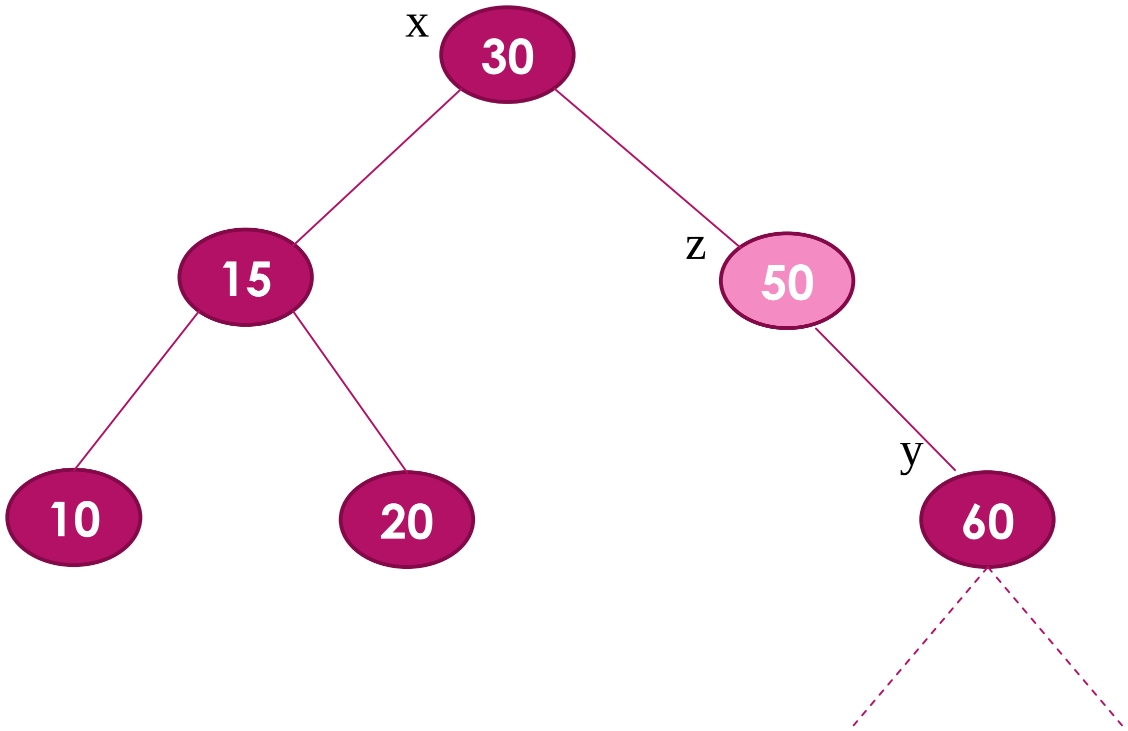


`z: internal node with a single child`

`The lone child y can replace z`

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

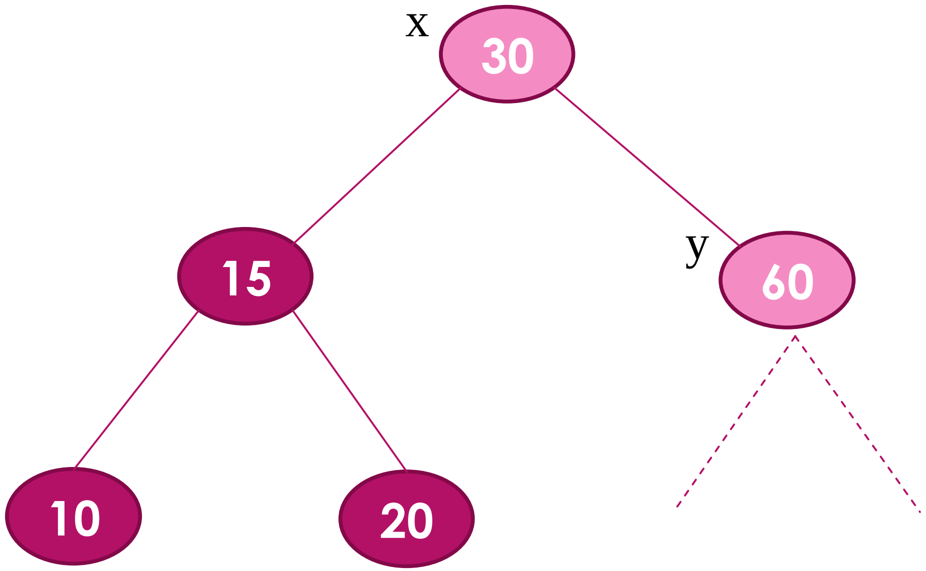




z: internal node with a single nonempty subtree

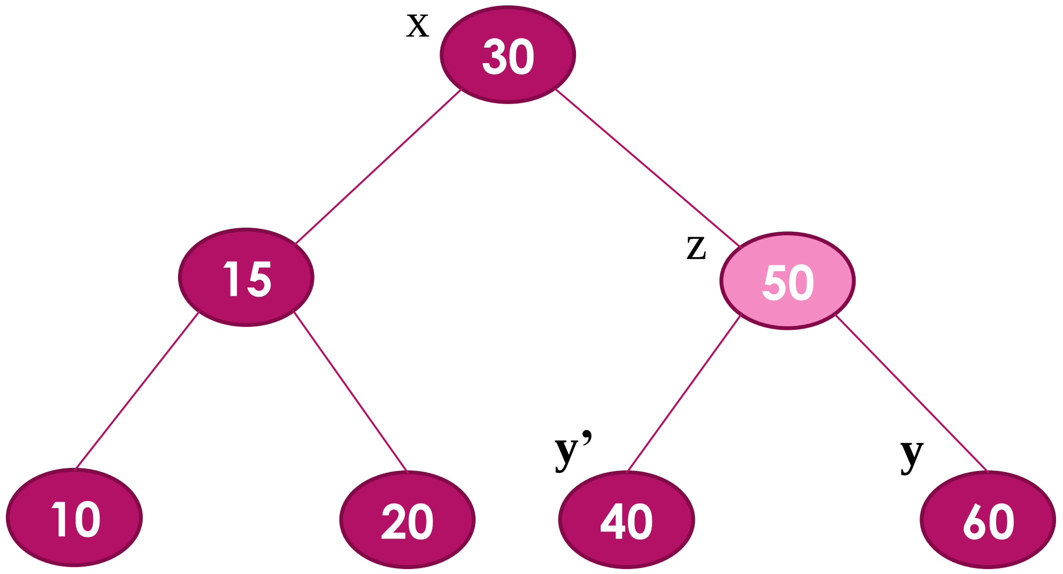
y can replace z

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



BST Deletion - examples

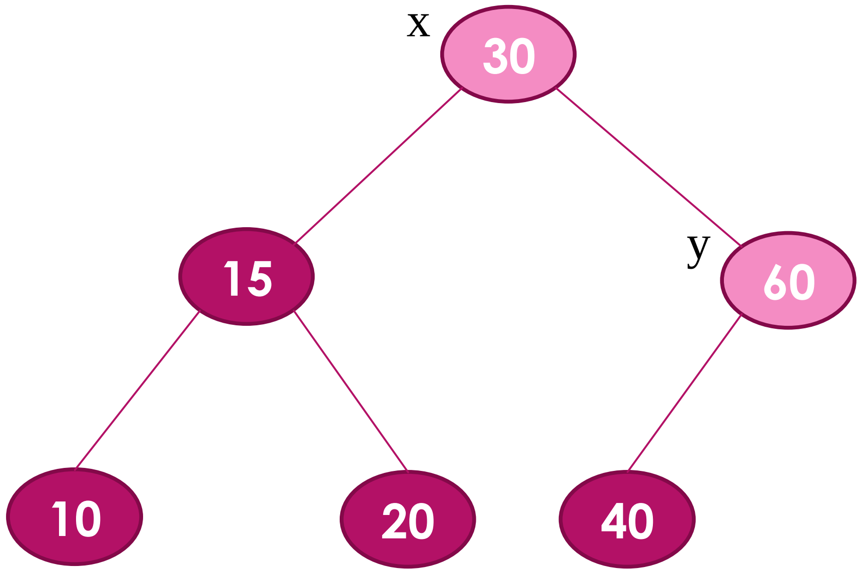
- Deletion of
  - a leaf node
  - a node with one child
  - ……………



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

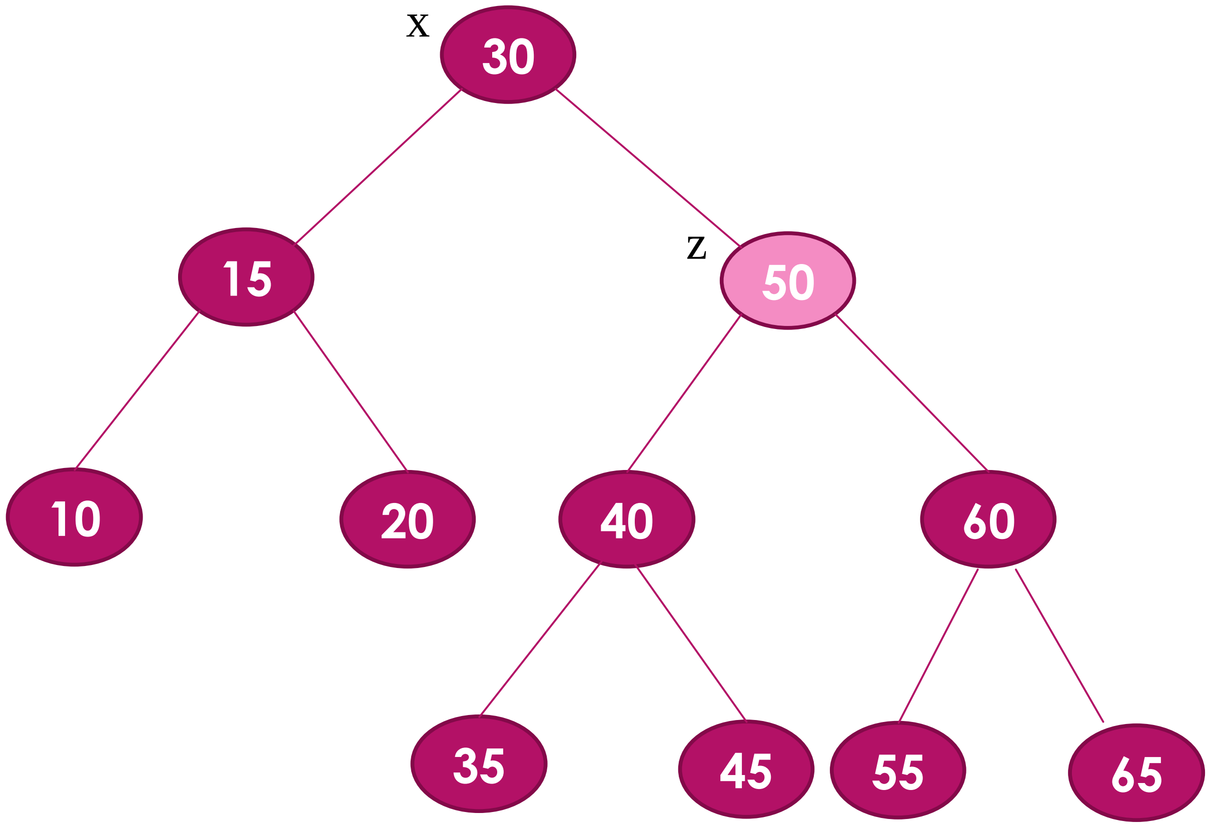
y.parent to be set as x
x.rchild to be set as y

Alternatively can y' replace z ?



BST Deletion - examples

- Deletion of
  - a leaf node
  - a node with one nonempty subtree
  - a node with both the children

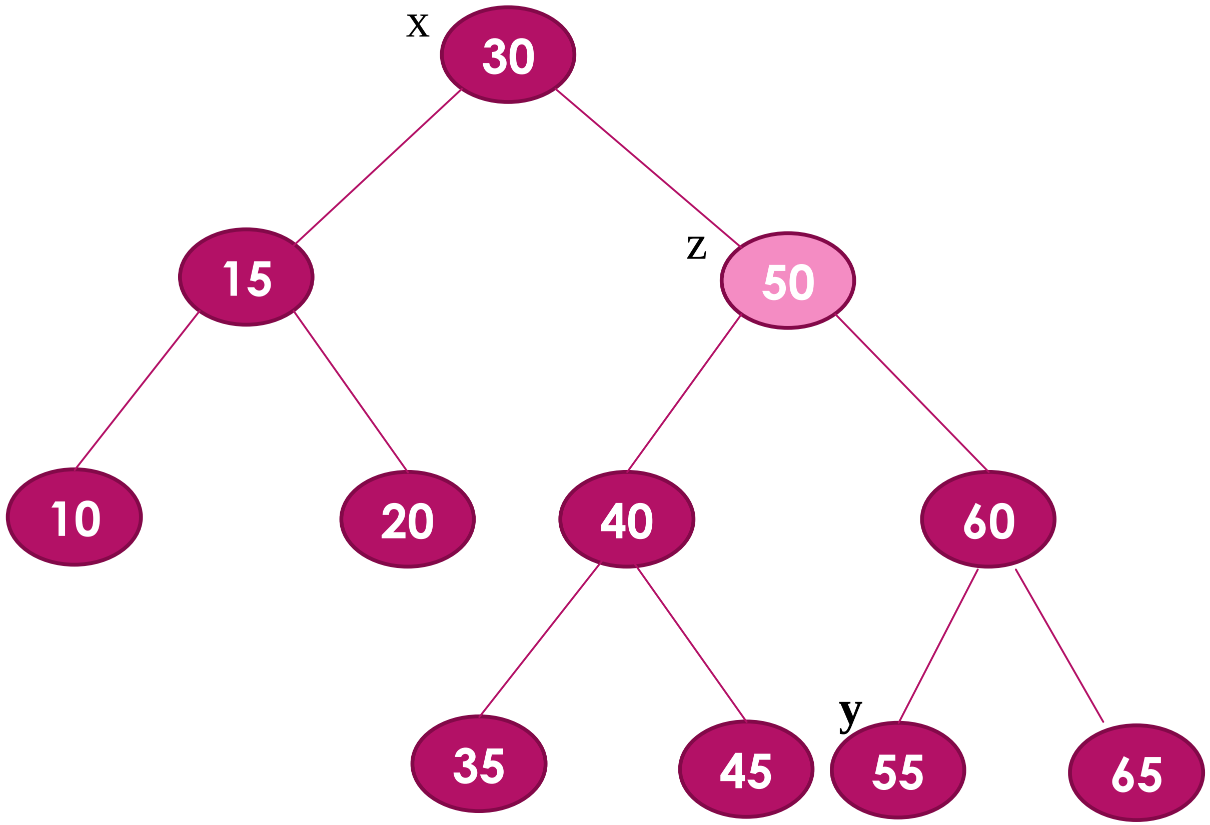


z: internal node, both the subtrees nonempty

Can not simply remove z

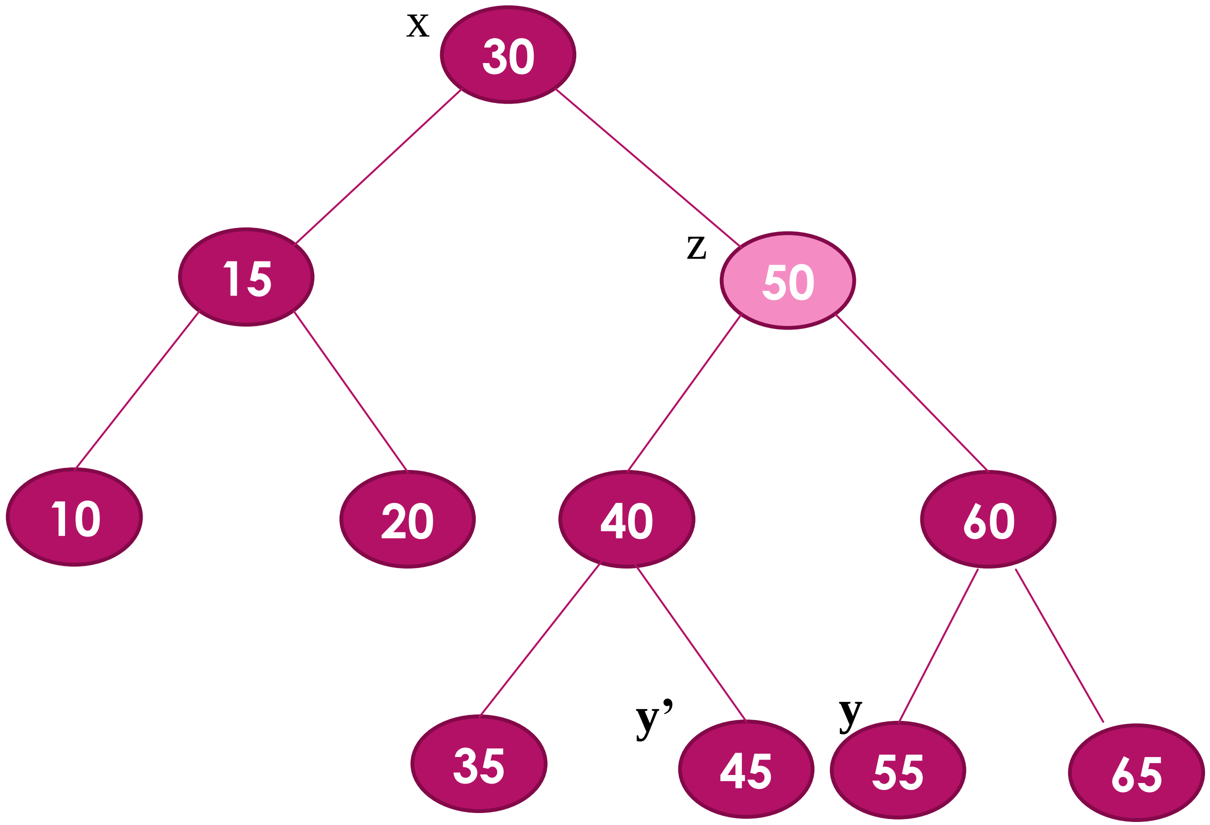
can z be replaced by some other node y ?

Can we select a y from the subtree of z?



z: internal node, both the subtrees nonempty

can z be replaced by y ?

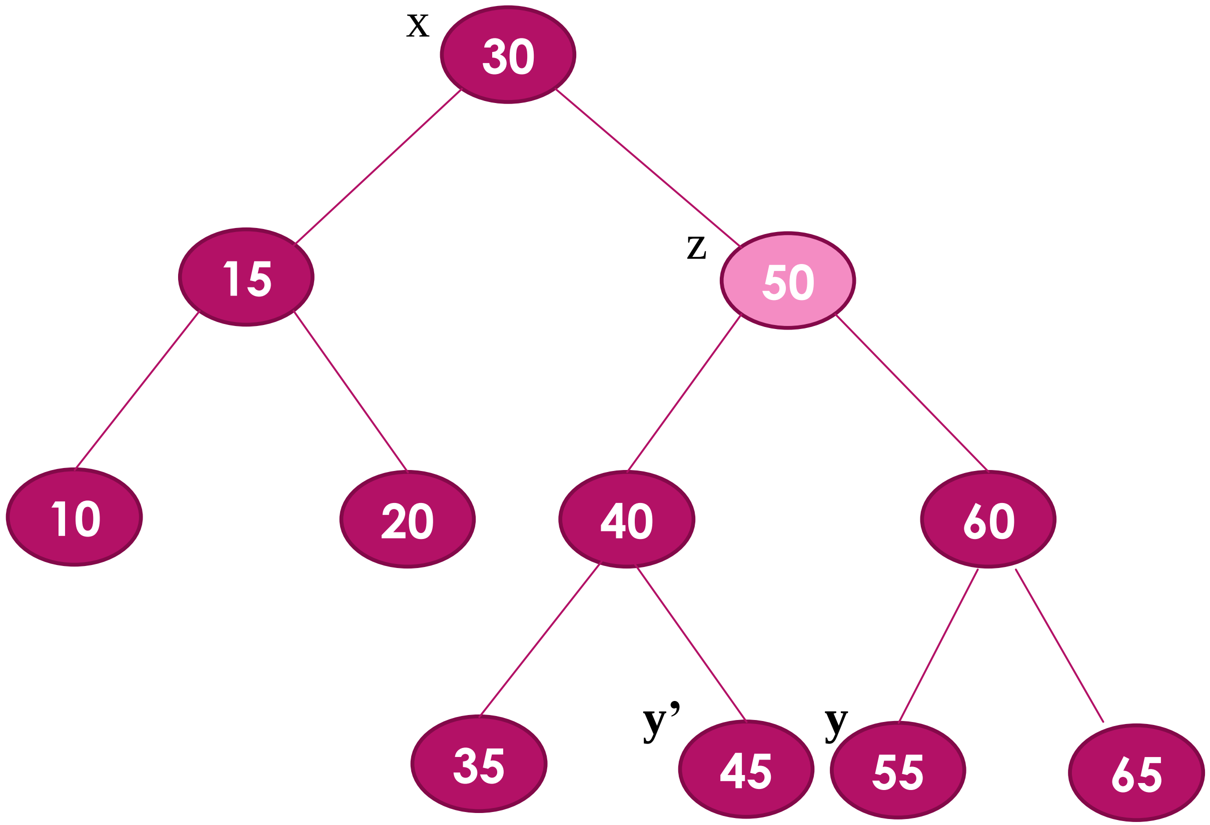


z: internal node, both the subtrees nonempty

can z be replaced by y′?

Replacing with z′s inorder successor/inorder predecessor

BST property is to be maintained



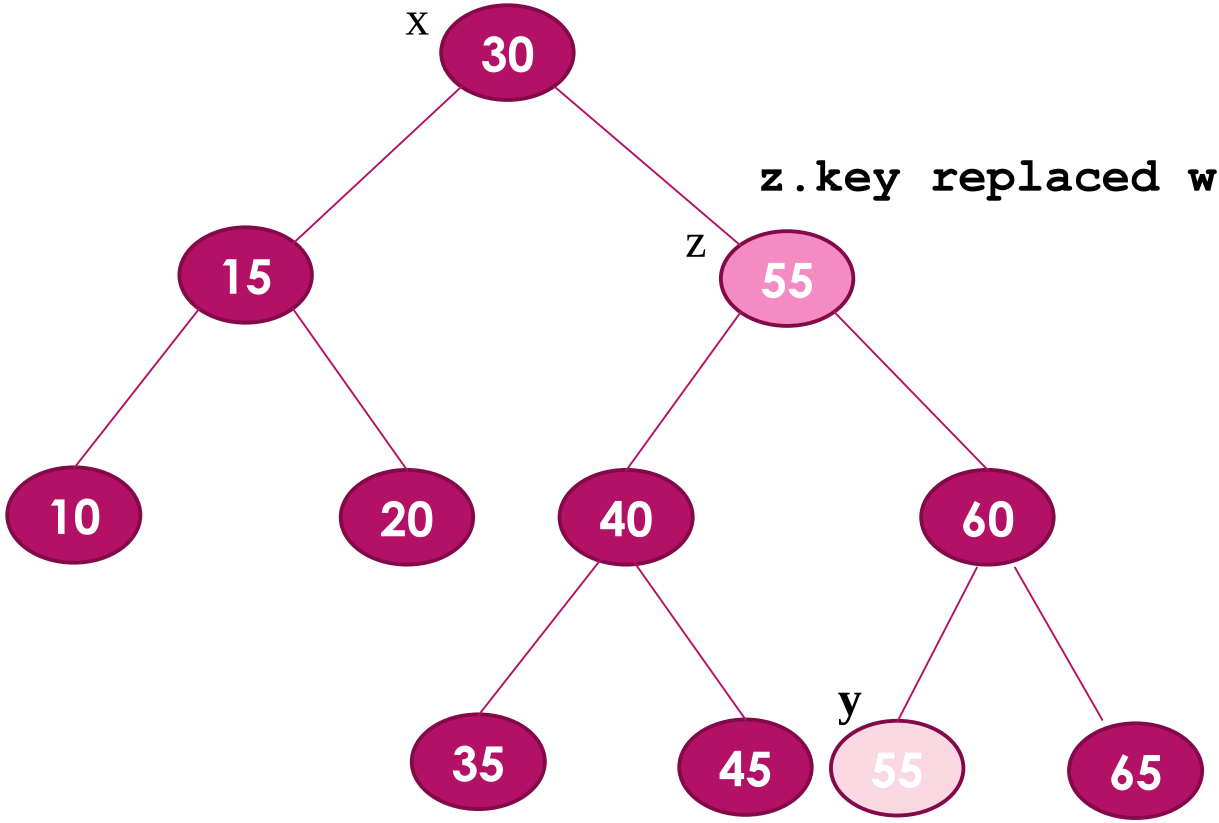
z: internal node, both the subtrees nonempty

Solution1:

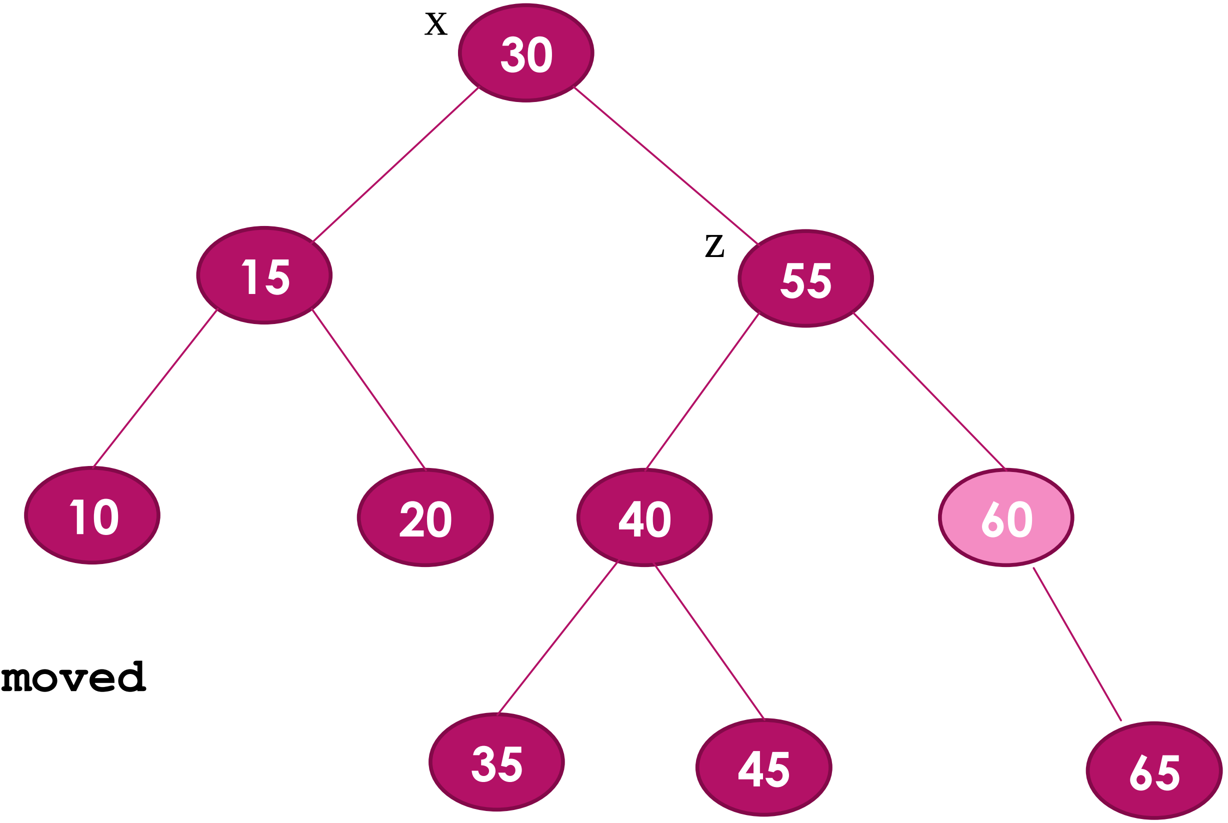
- Let y be z′s inorder successor
- Replace z.key with y.key
- Delete y

z is not physically removed

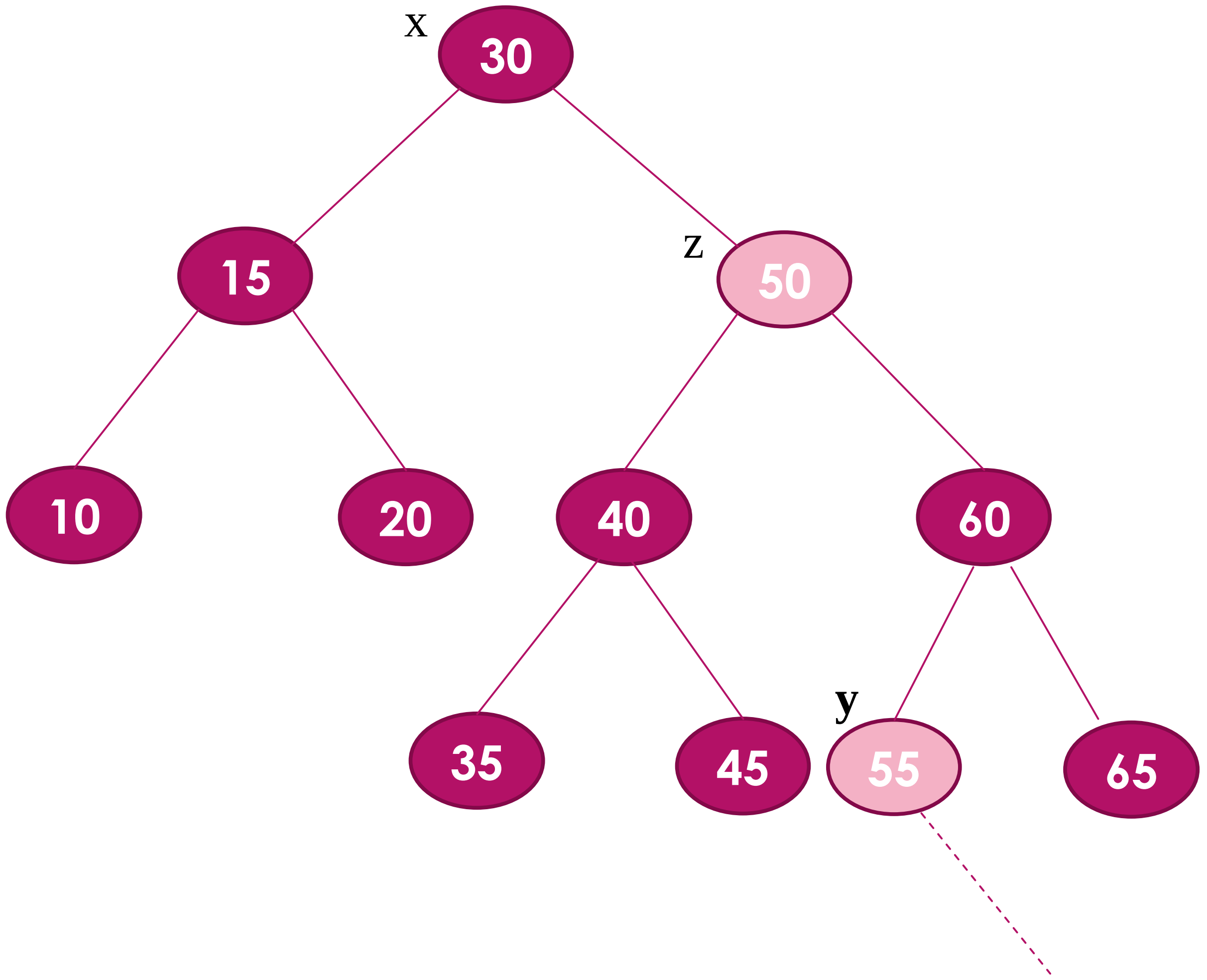
Is it correct to replace z.key with y′.key, where y′ is the inorder predecessor of z and then remove y′?



z.key replaced with y.key



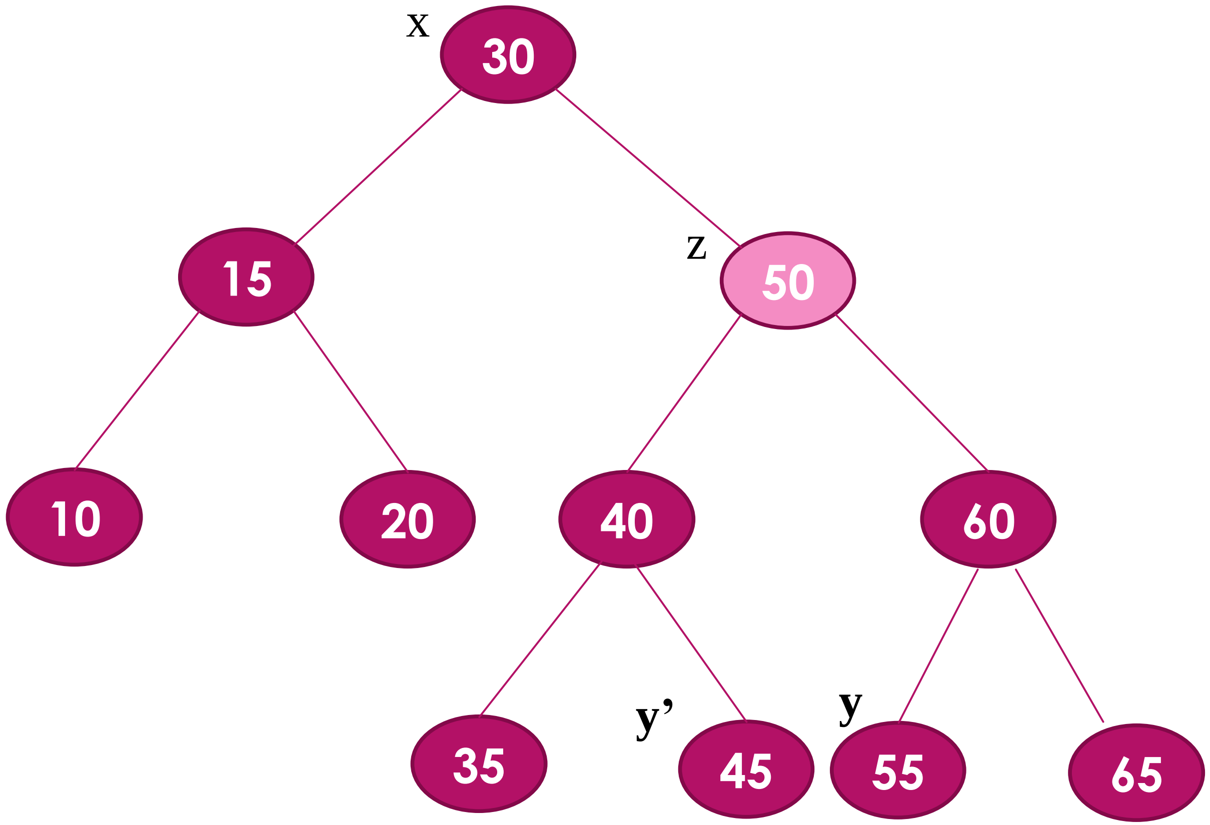
y physically removed



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.):

- 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 - examples

- Deletion of
  - a leaf node
  - a node with one nonempty subtree
  - a node with both the subtrees non empty

BST Deletion - Cases

- Let z be the node to be deleted

1. z has no children
2. z has just one child
3. z has both left and right child

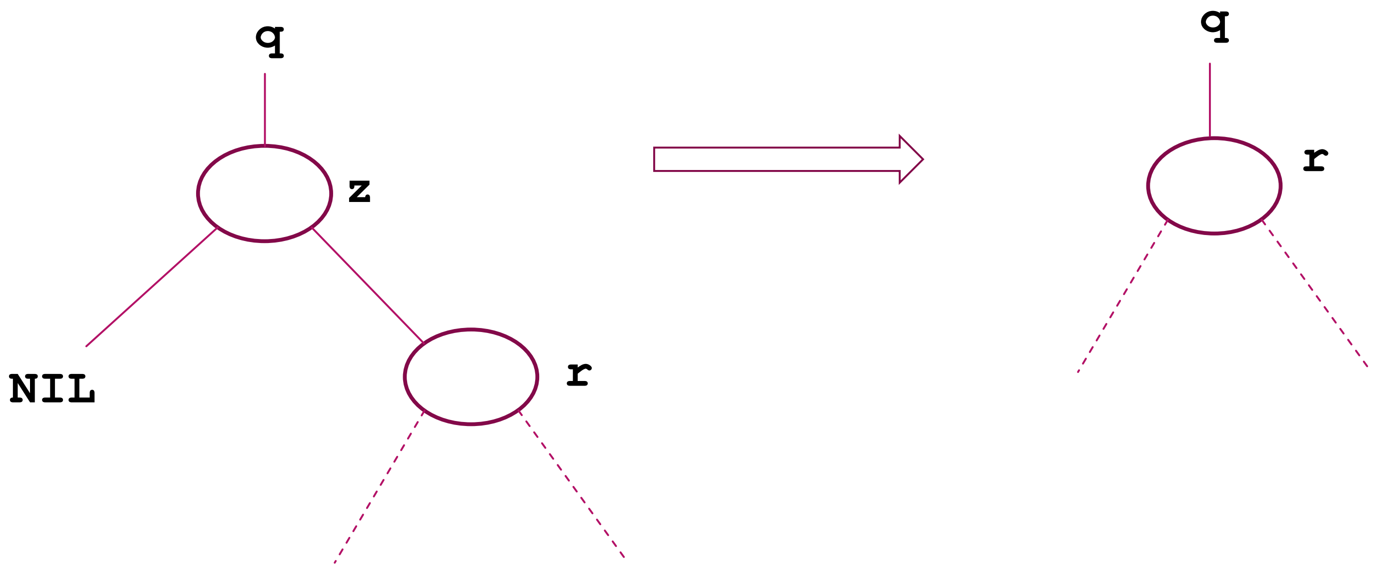
BST Deletion - Algorithm : Cases

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

1. **z has no left child**
2. **z has just one child, which is its left child**
3. **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)

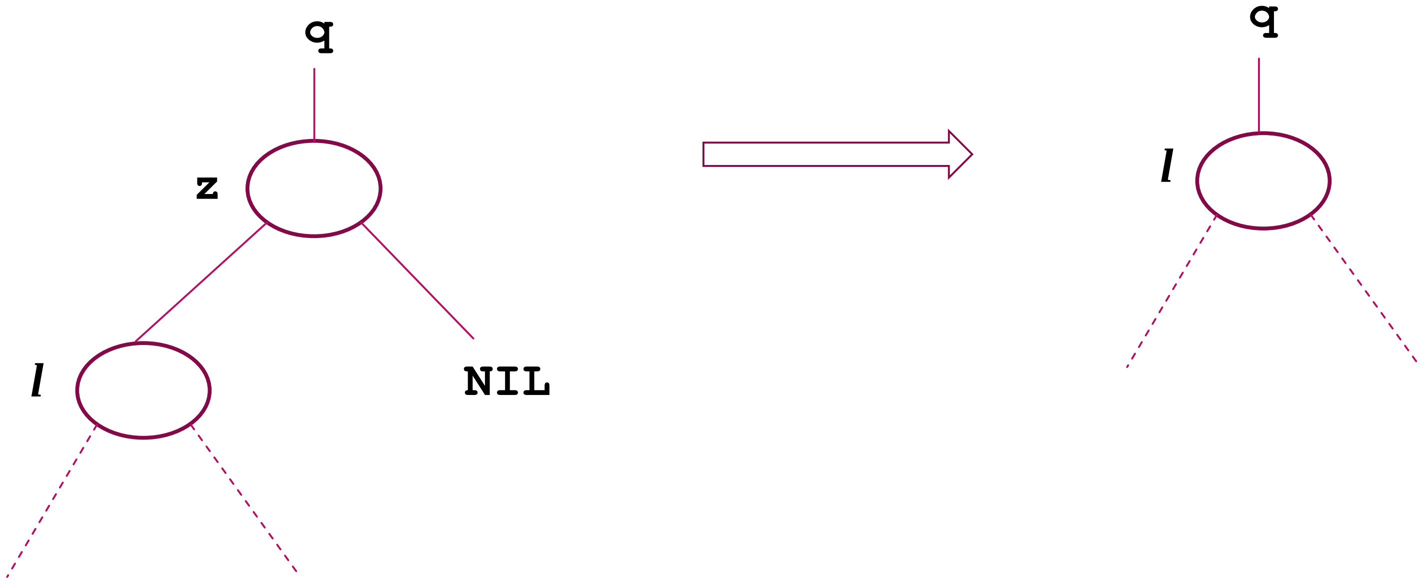


z has no left child

Replace z by its right child r

r may or may not be empty

(b)

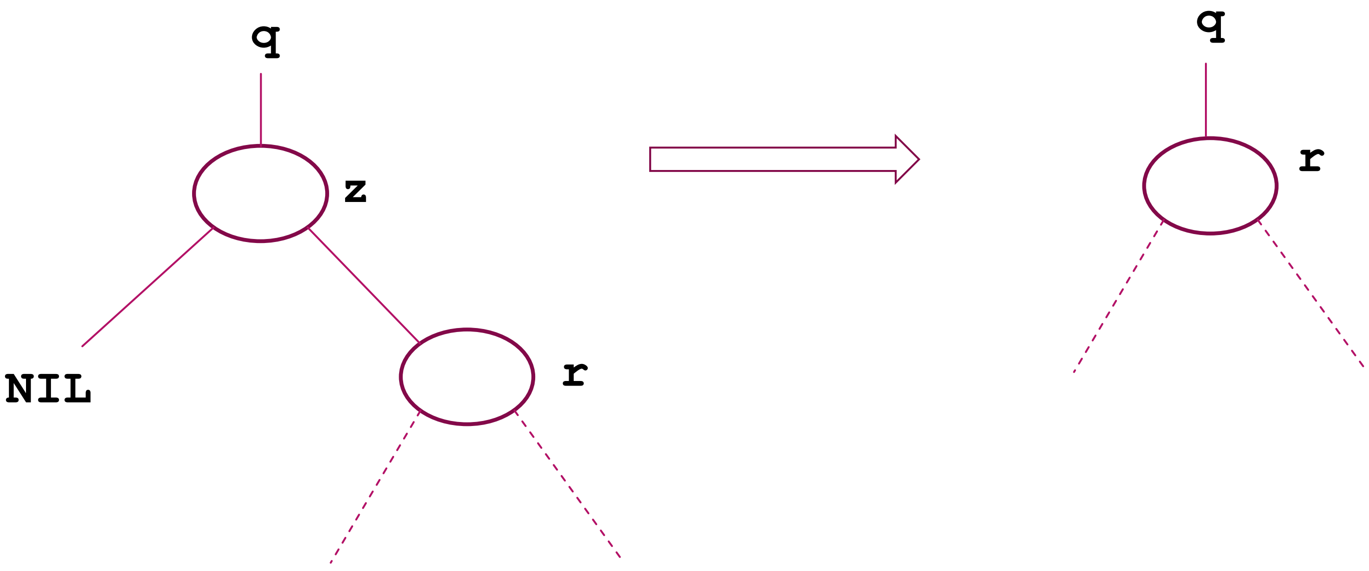


z has left child *l* no right child
Replace z by its left child

BST Deletion : Cases a and b

```
TREE-DELETE(T, z)
   if z.left == NIL
       TRANSPLANT (T, z, z.right)
   elseif z.right == NIL
       TRANSPLANT (T, z, z.left)
   else ……… //z has both children, split into two cases c and d
```

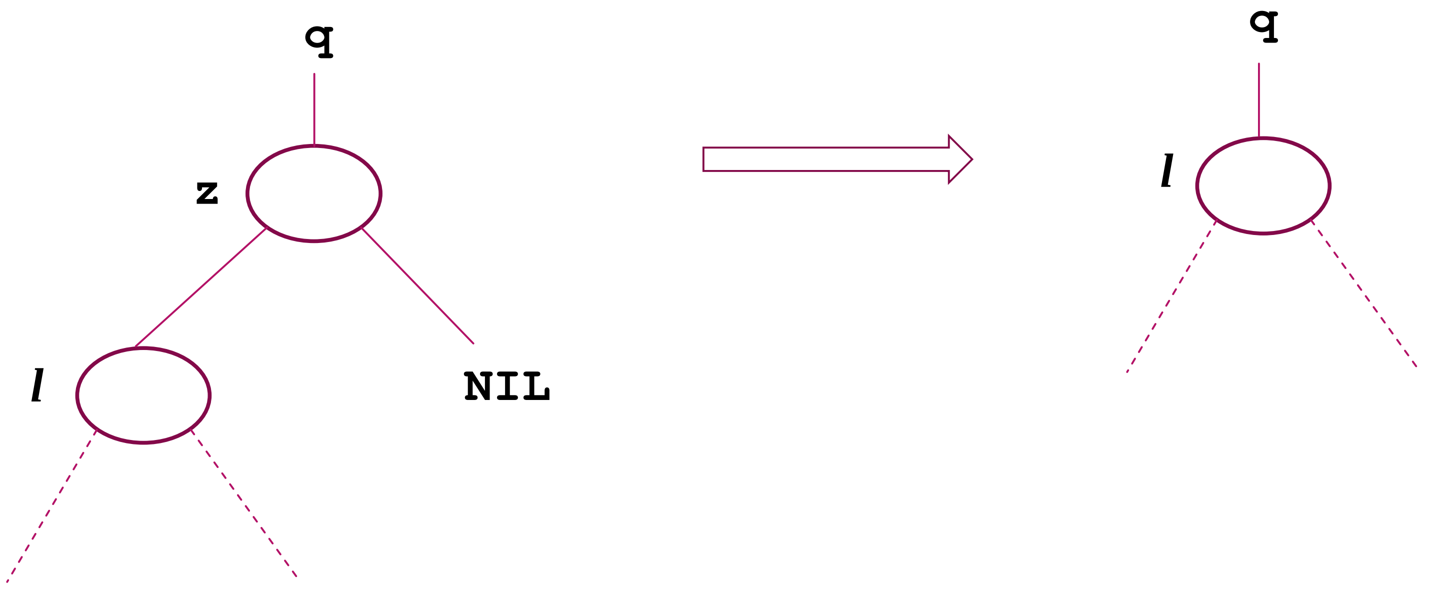
(a)



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

subtree rooted at z replaced with the subtree rooted at r

(b)



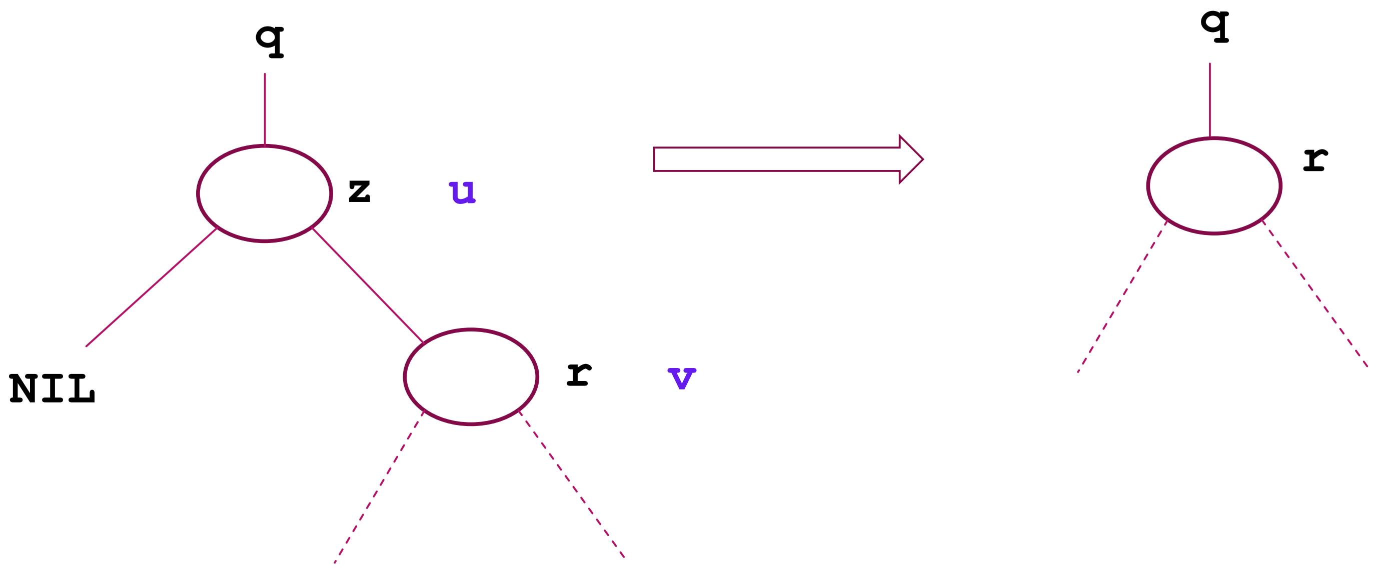
```
…….elseif z.right == NIL
           TRANSPLANT (T, z, z.left) //z has left child l
```

subtree rooted at z replaced with the subtree rooted at *l*

BST Deletion : Cases a and b

```
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
                   // to be done by the caller
```

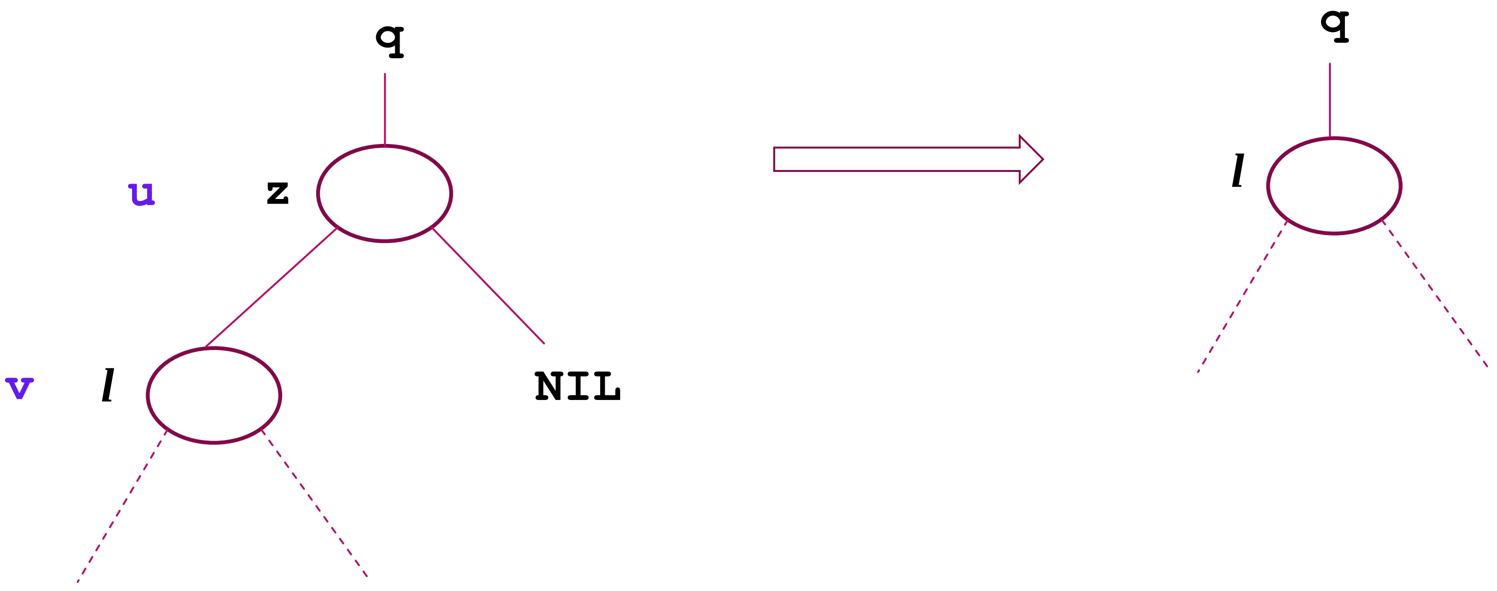
(a)



```
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

(b)



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

Reference

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