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)

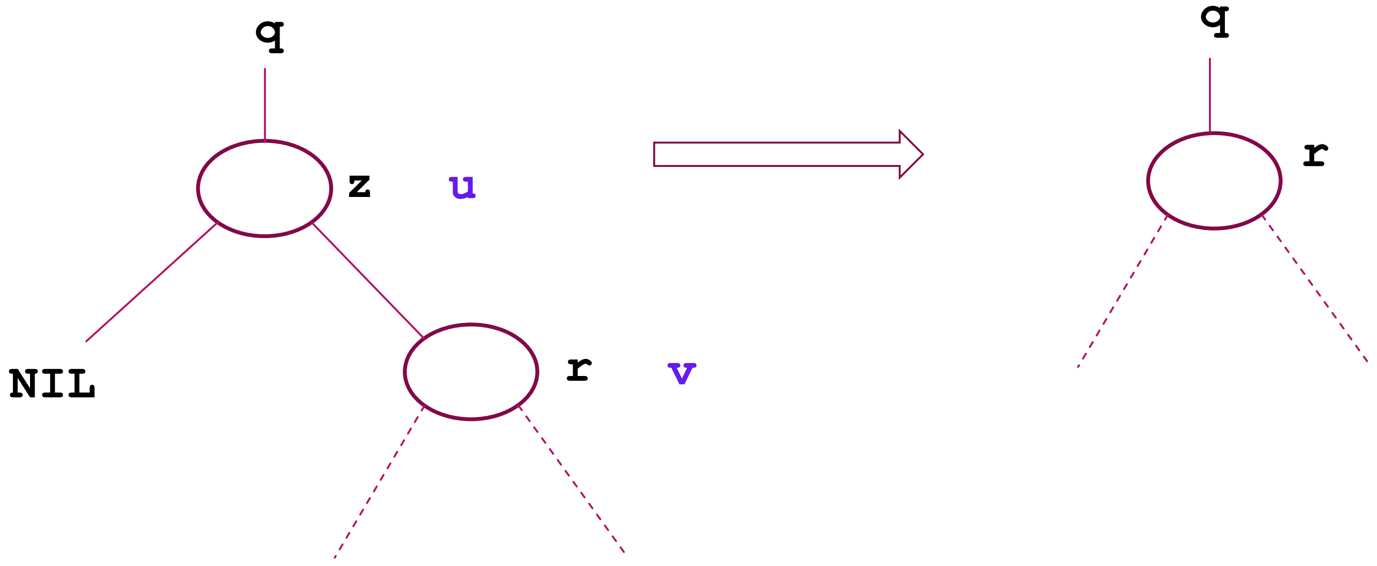
BST Deletion : Cases a and b

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

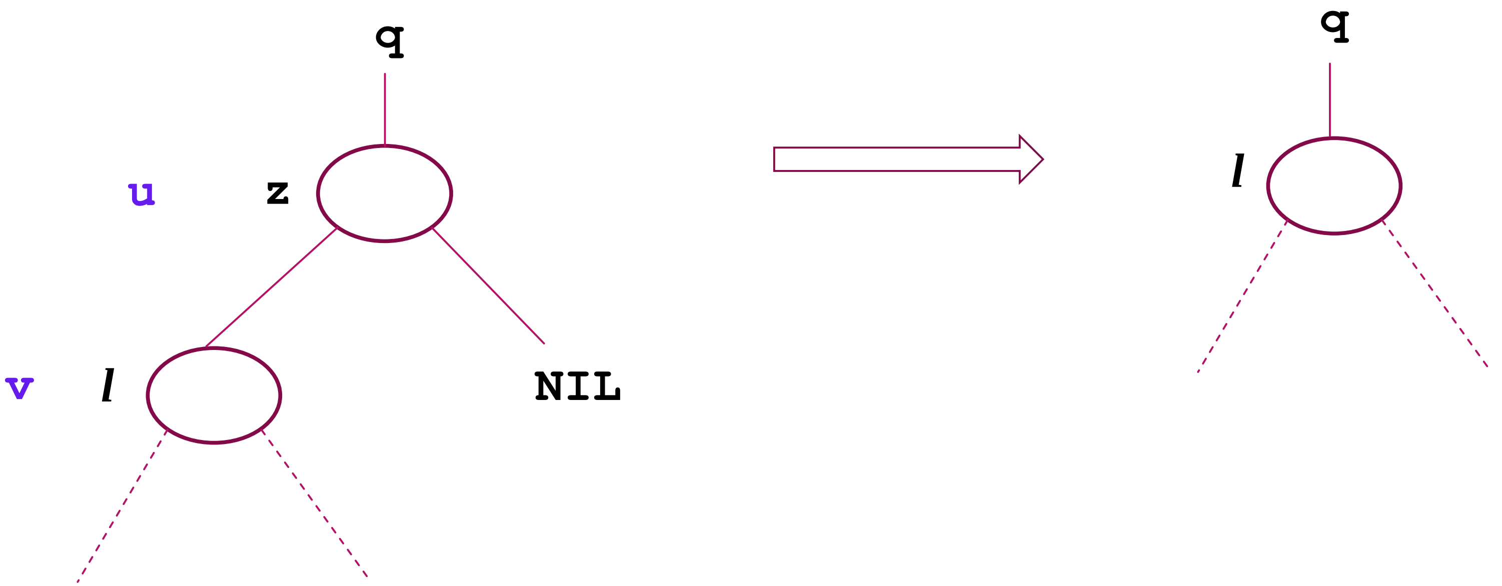
(a)



```
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

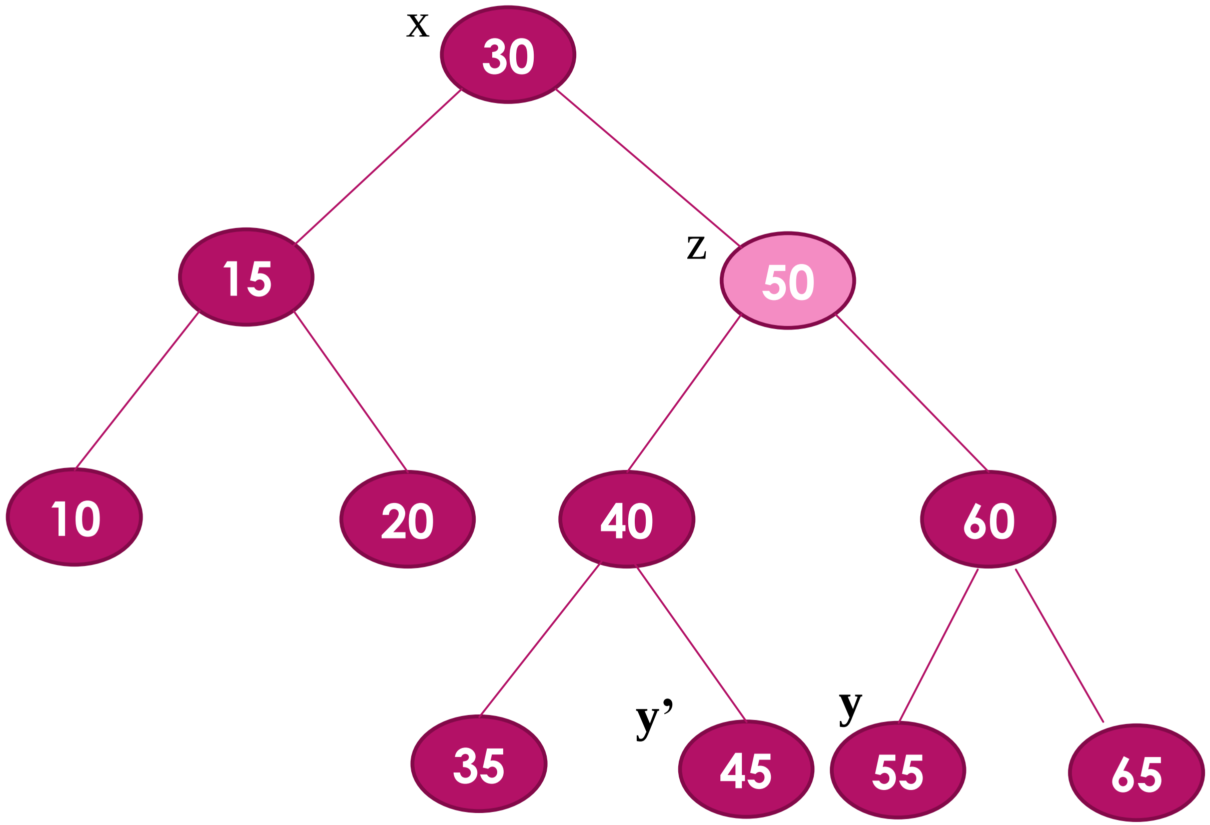
(b)



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

BST Deletion - Algorithm : Cases

- **The node to be deleted has both left child and right child**



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: 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
```

BST Deletion : Cases c and d

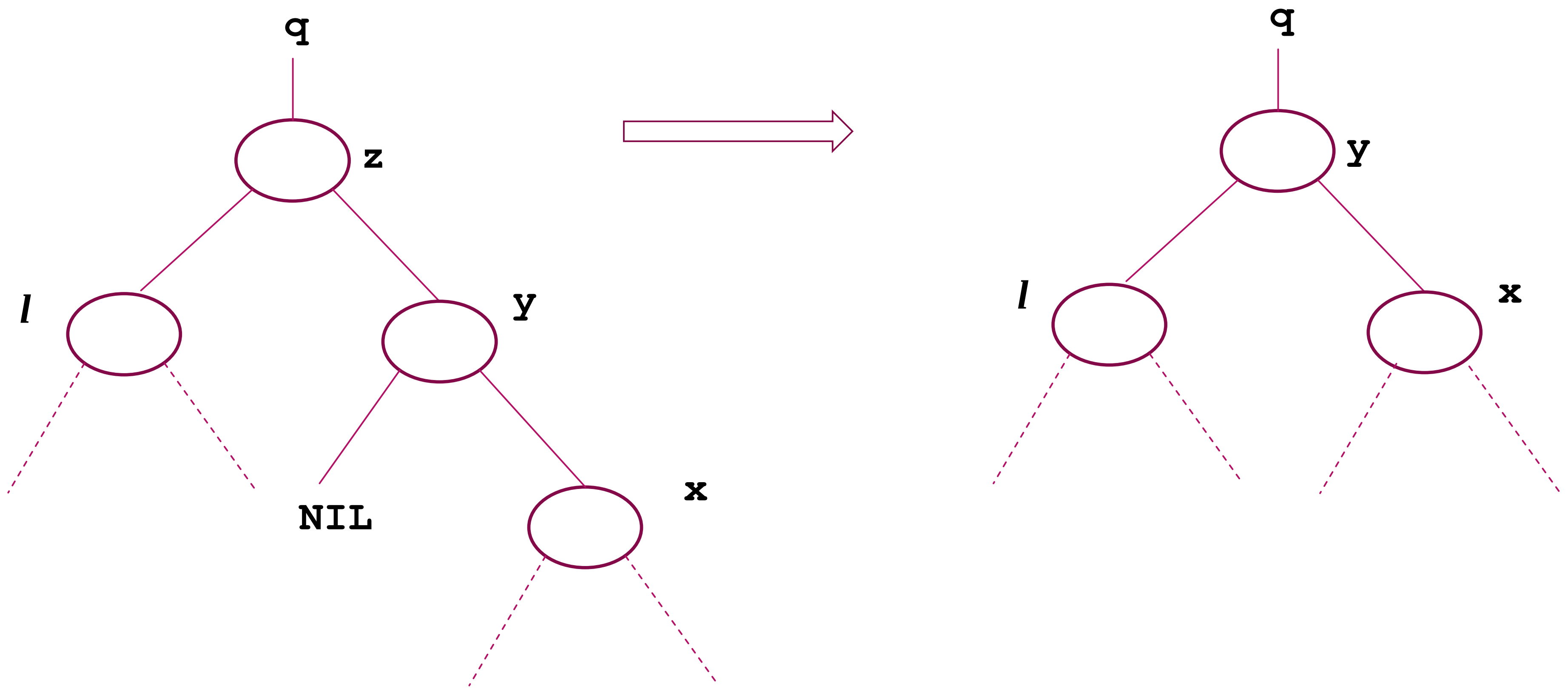
z has both a left and a right child

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

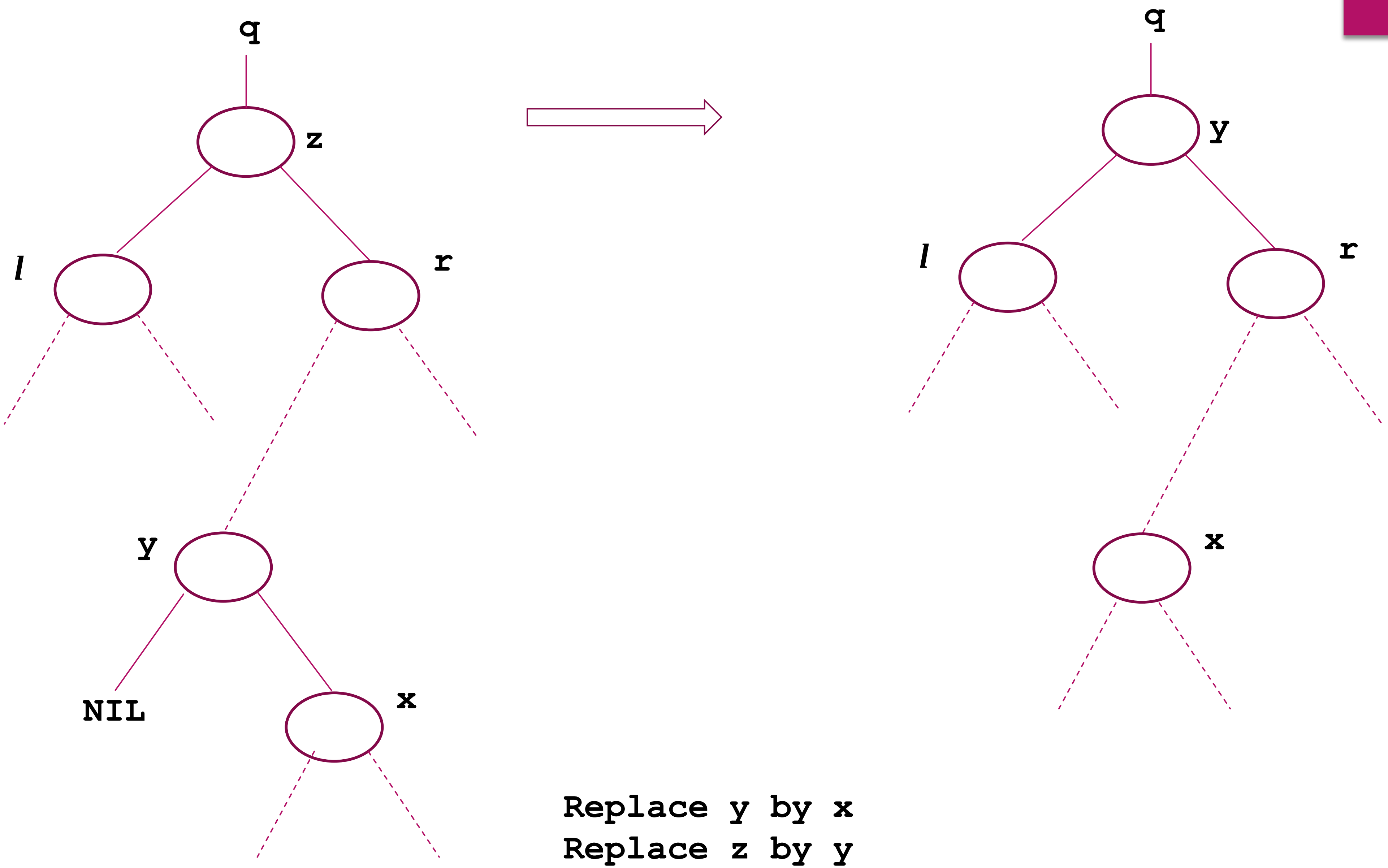
(c) Successor y is z’s right child



```
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 ……
```

(d) Successor y is not right child of z, y lies in the subtree rooted at r(r is z's right child)



Replace y by x
Replace z by 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
              y.right.p = y                                    (d)
       TRANSPLANT (T, z, y)
       y.left = z.left                  (c)
       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
            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**