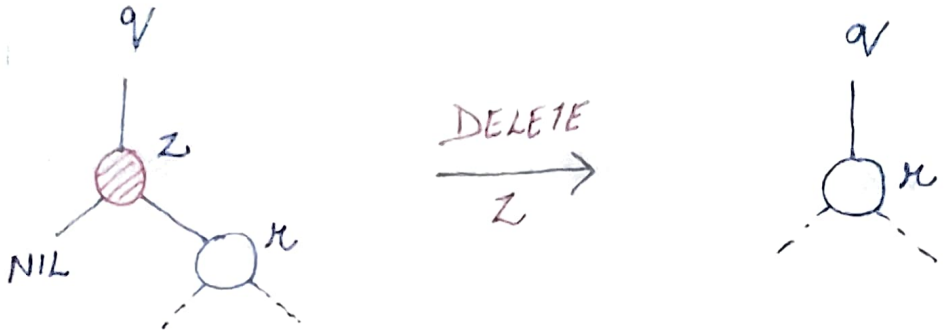
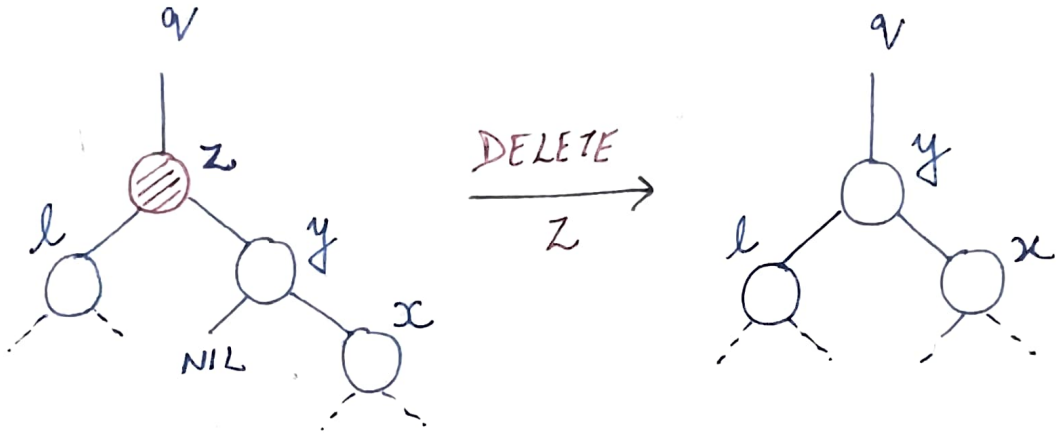
* BST DELETION : 4 CASES

(1)

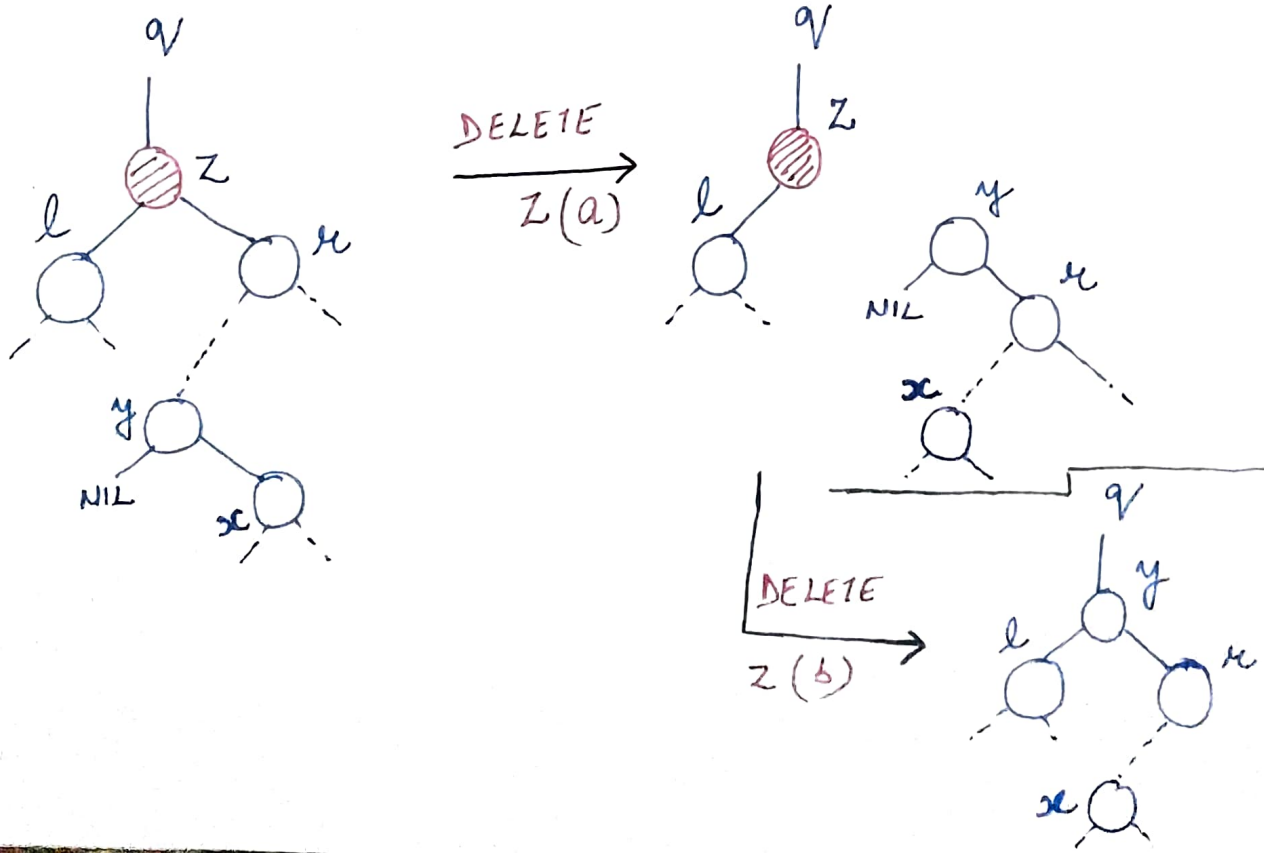


(2)

(3)



(4)



* BST DELETION :

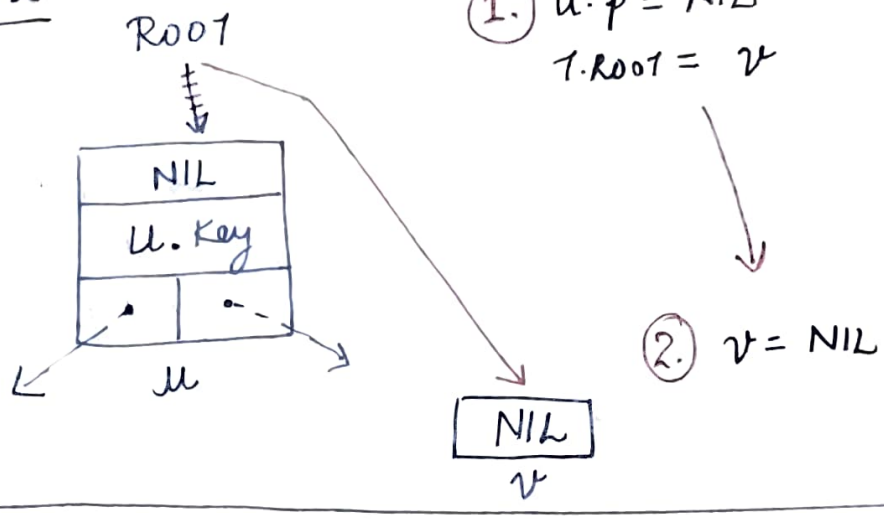
→ TRANSPLANT Algorithm : In BST (T), the TRANSPLANT Algo replaces the subtree rooted at u with the subtree rooted at v.

---

* CASE : ROOT NODE IS BEING DELETED

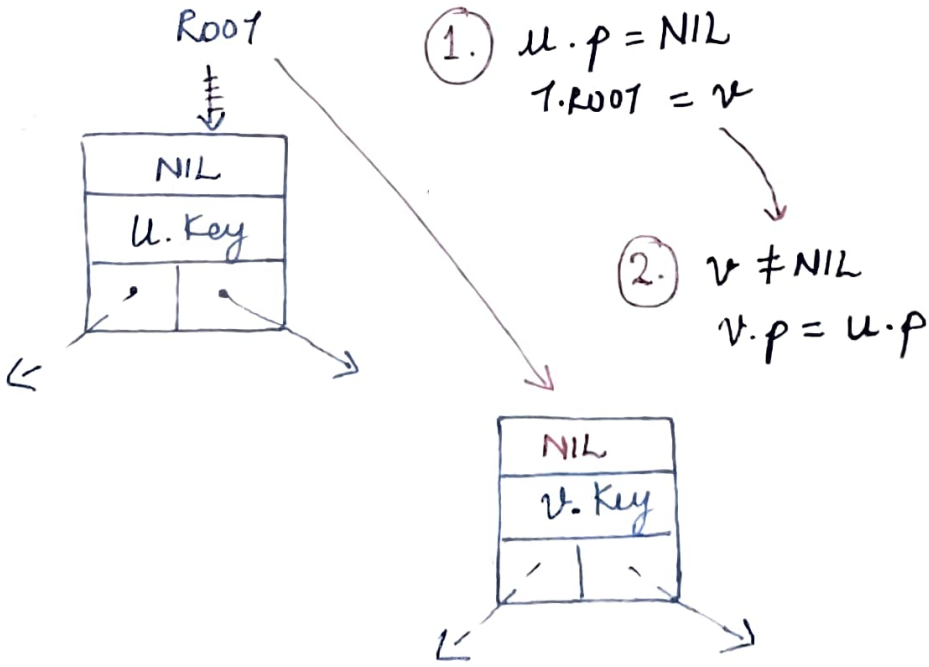
TRANSPLANT(T, u, v)

(a) v = NIL



1. $u.p$ = NIL
   $T.ROOT = v$
2. v = NIL

(b) $v \neq$ NIL

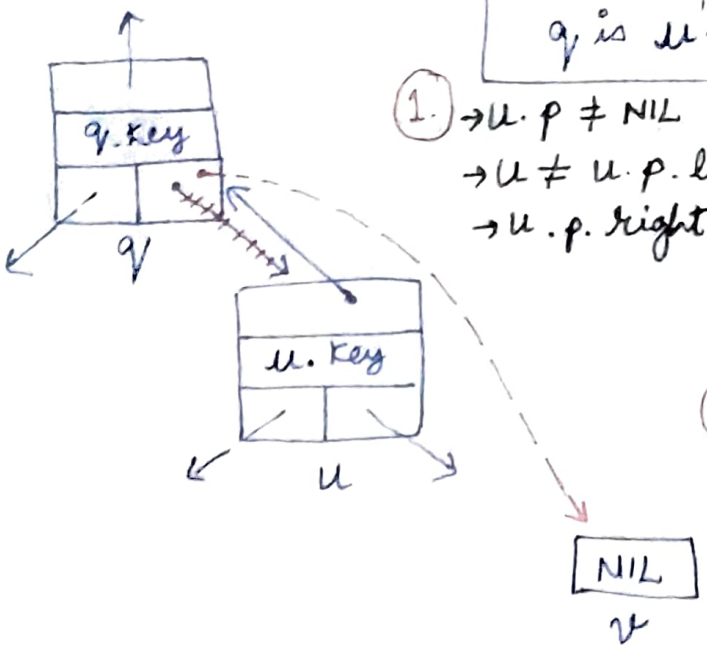


1. $u.p$ = NIL
   $T.ROOT = v$
2. $v \neq$ NIL
   $v.p = u.p$

* GENERAL CASE :

TRANSPLANT (T, u, v)
q is u's parent

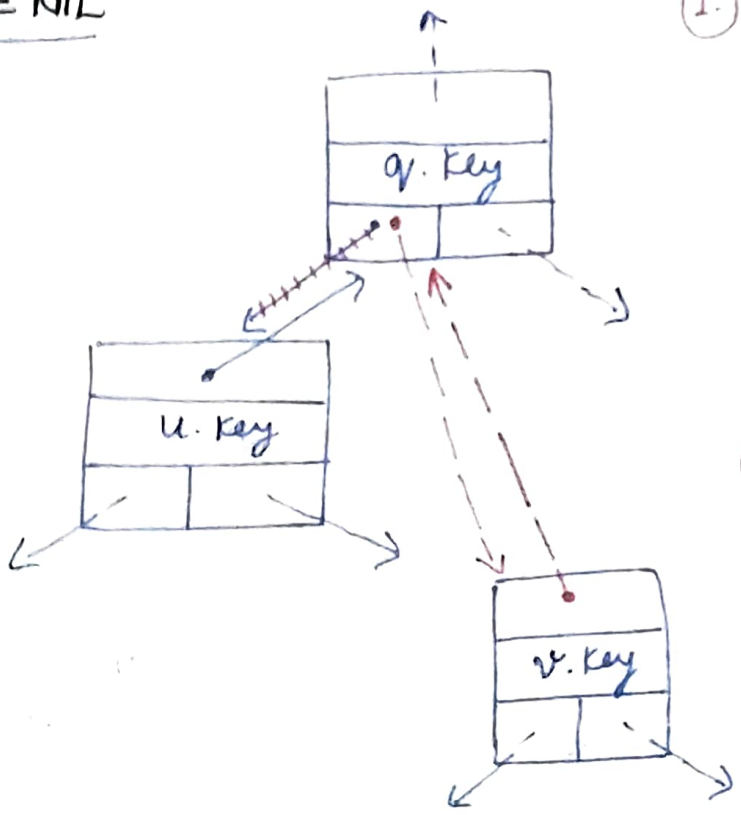
(a) v = NIL



1. → u.p ≠ NIL
→ u ≠ u.p.left
→ u.p.right = v

2. v = NIL

(b) v ≠ NIL



1. → u.p ≠ NIL
→ u == u.p.left
→ u.p.left = v

2. v ≠ NIL
v.p = u.p

NOTE :

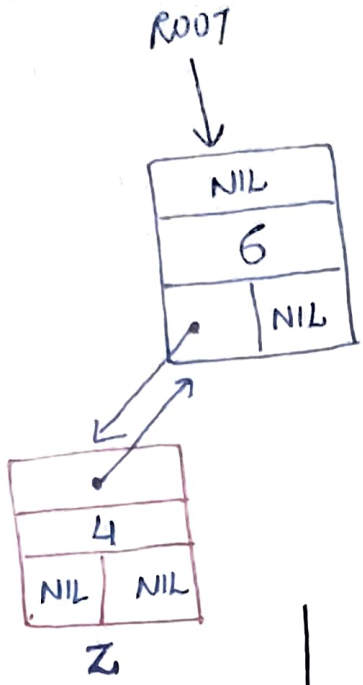
TRANSPLANT Algo does not attempt its update v.left and v.right ; doing so and not doing so, is the responsibility of TRANSPLANT's caller function.

* TREE - DELETE (T, z)

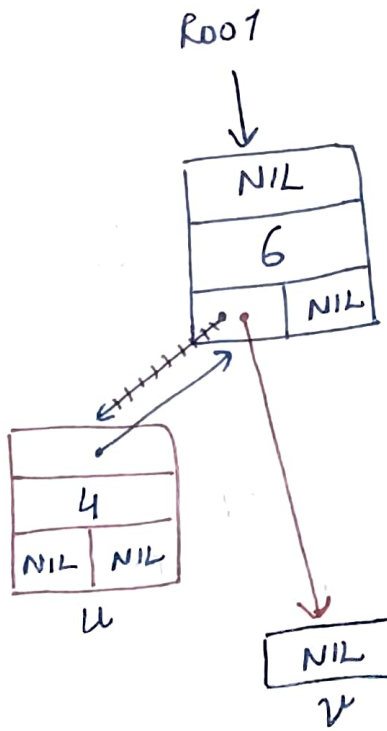
→ we need to delete z node from Tree (BST) T.

---

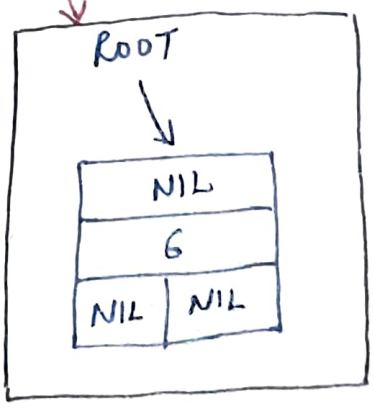
(1)



1. → z. left == NIL
   → TRANSPLANT (T, z, z. right)
   [TRANSPLANT (T, z, NIL)]
   u v

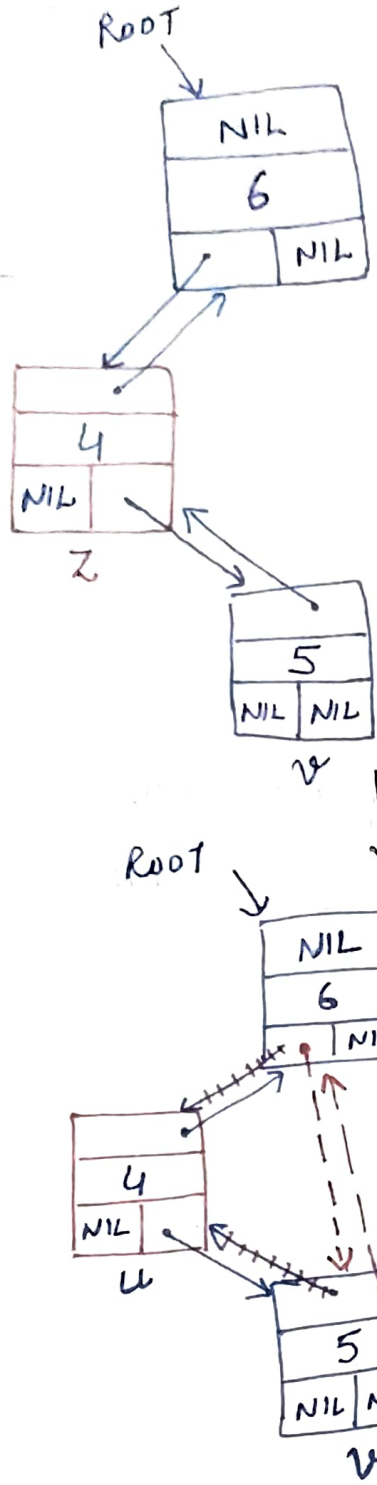


2. → u.p ≠ NIL
   → u == u.p.left
   → u.p.left = v



FINAL OUTCOME

(2.)



Key value of z is 4

(1.) TREE-DELETE (T, z)

→ z.left = NIL

→ TRANSPLANT (T, z, z.right)

[TRANSPLANT (T, z, z.right)]

[having key value 5.]

(2.) → u.p ≠ NIL

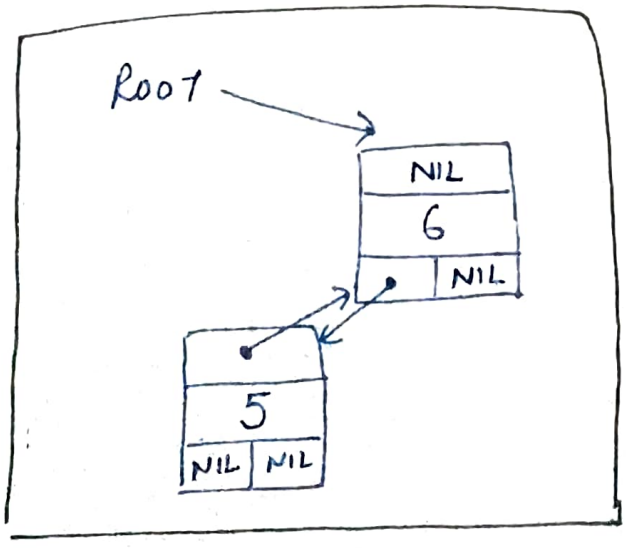
→ u = u.p.left

→ u.p.left = v

(3.) → v ≠ NIL

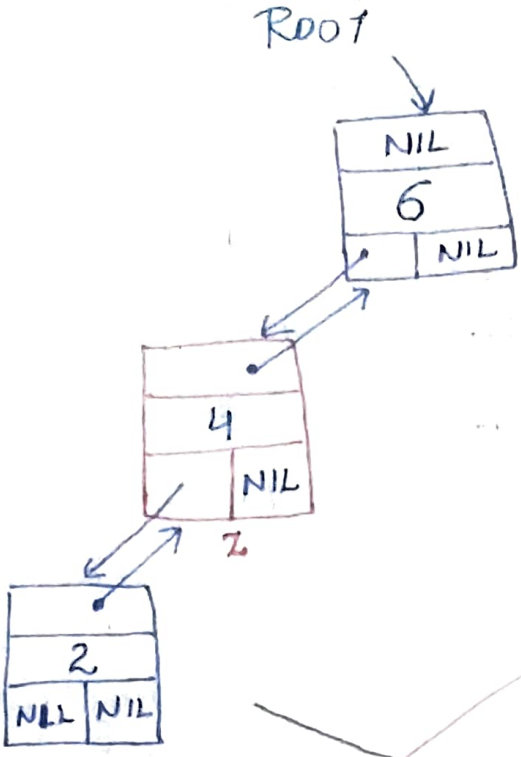
→ v.p = u.p

FINAL OUTCOME



This will be the final tree after deletion of node having key value 4.

(3).



TREE-DELETE (T, z)

(1.) → z.left ≠ NIL

→ z.right == NIL

→ TRANSPLANT (T, z, z.left)

[TRANSPLANT (T, z, z.left)]

↓ key value 2

key value 4

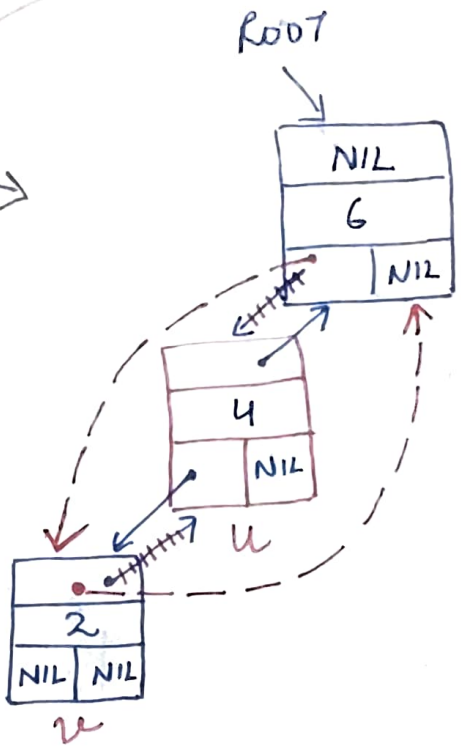
(2.) → u.p ≠ NIL

→ u == u.p.left

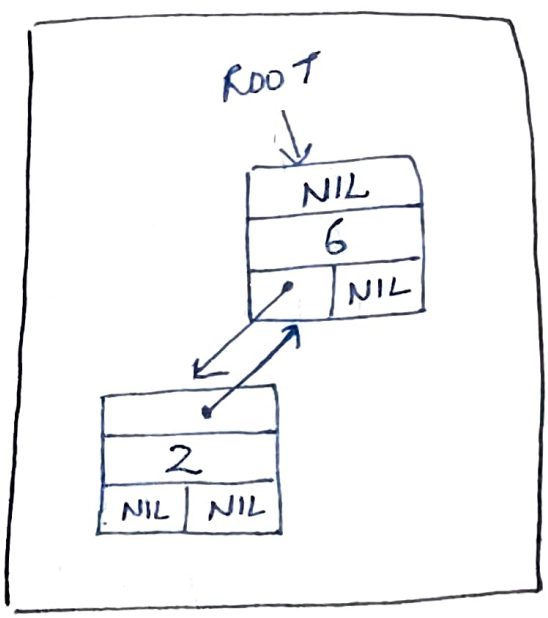
→ u.p.left = v

(3.) v ≠ NIL

v.p = u.p

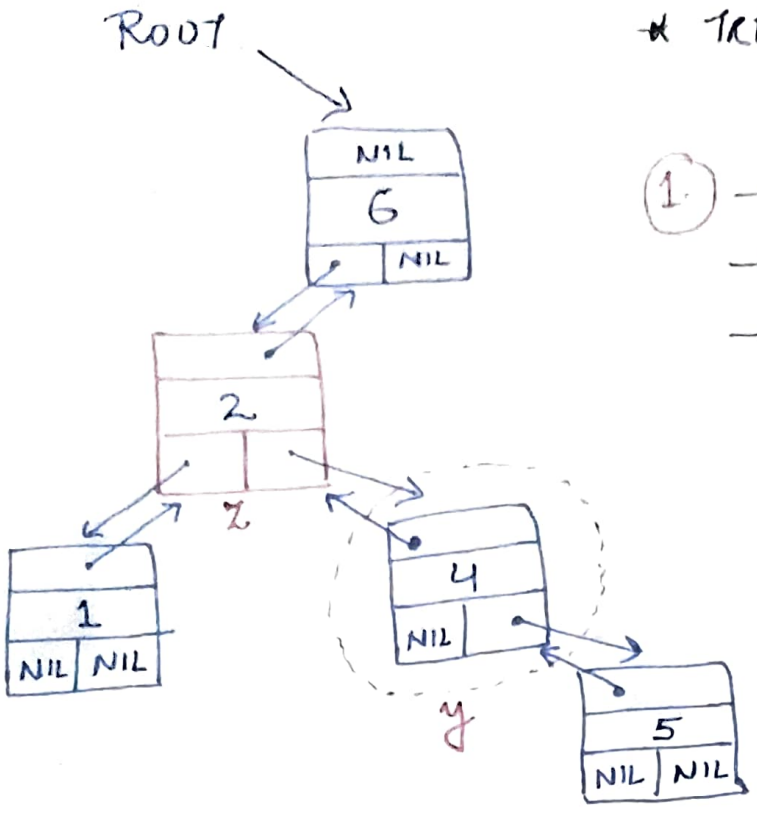


FINAL OUTCOME



This is the final tree after deletion of node having key value 4.

(4)



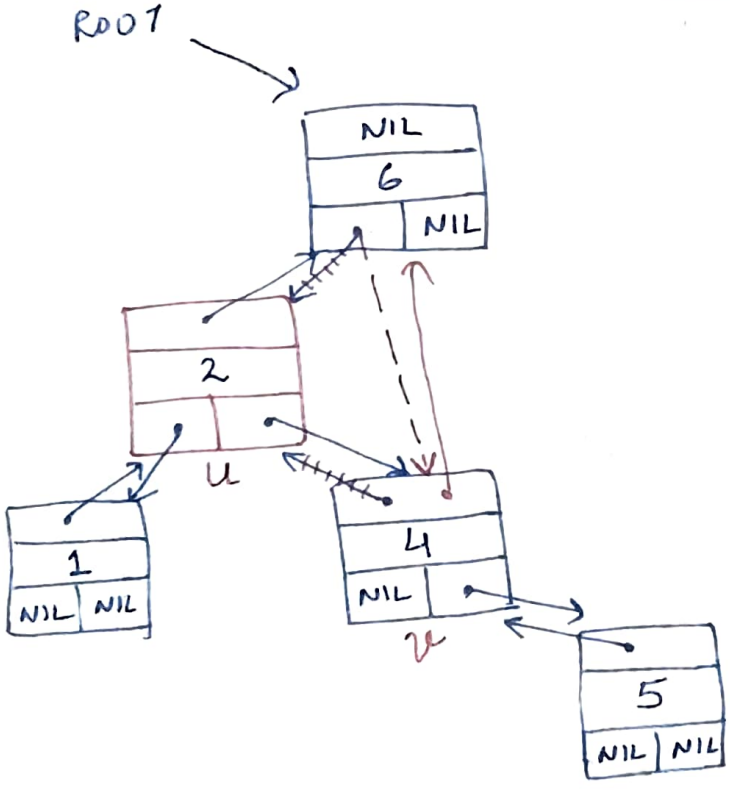
* TREE-DELETE (T, z)
(key value 2)

(1) → z.left ≠ NIL
→ z.right ≠ NIL
→ y = TREE-MINIMUM(z.right)
[z.right has key value 4]

TREE-MINIMUM (z.right)
returns node having key value 4

So, y is pointing to the node having key value 4.

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

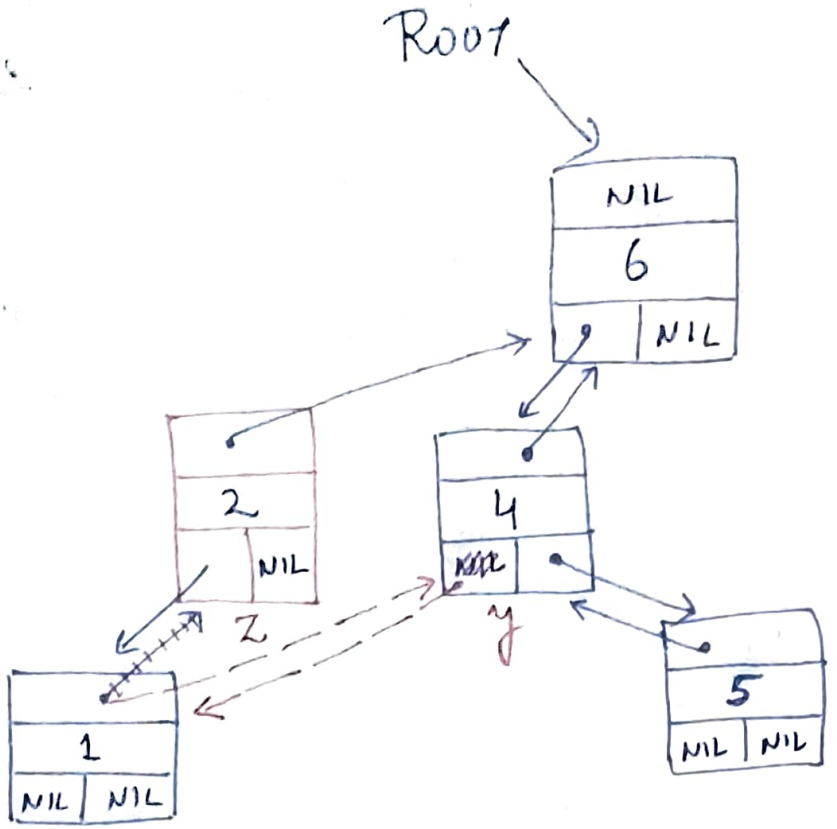


(3) → u.p ≠ NIL
→ u == u.p.left
→ u.p.left = v

(4) v ≠ NIL
v.p = u.p

* ON NEXT PAGE

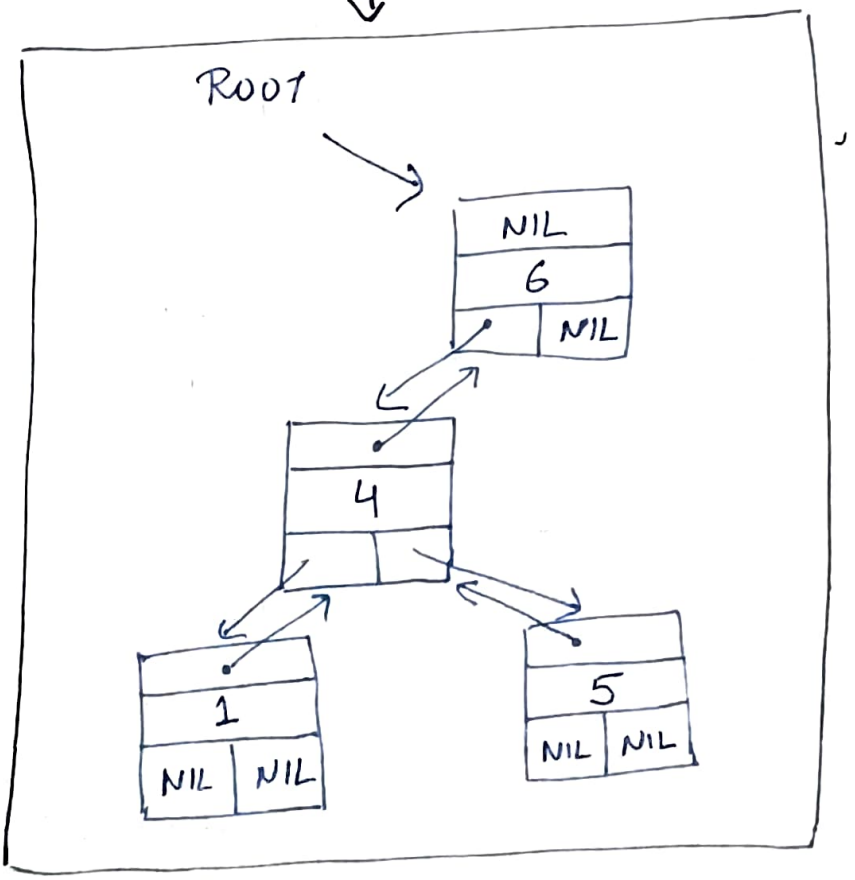
Contd.



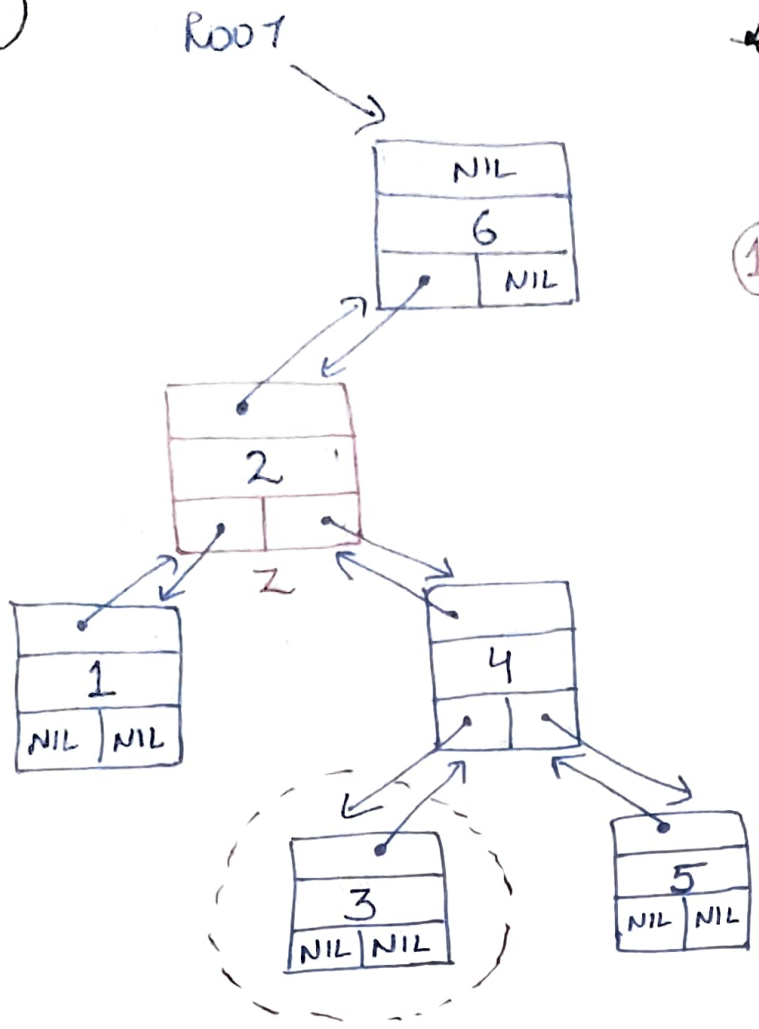
5. y.left = z.left

y.left.p = y

FINAL OUTCOME



5.

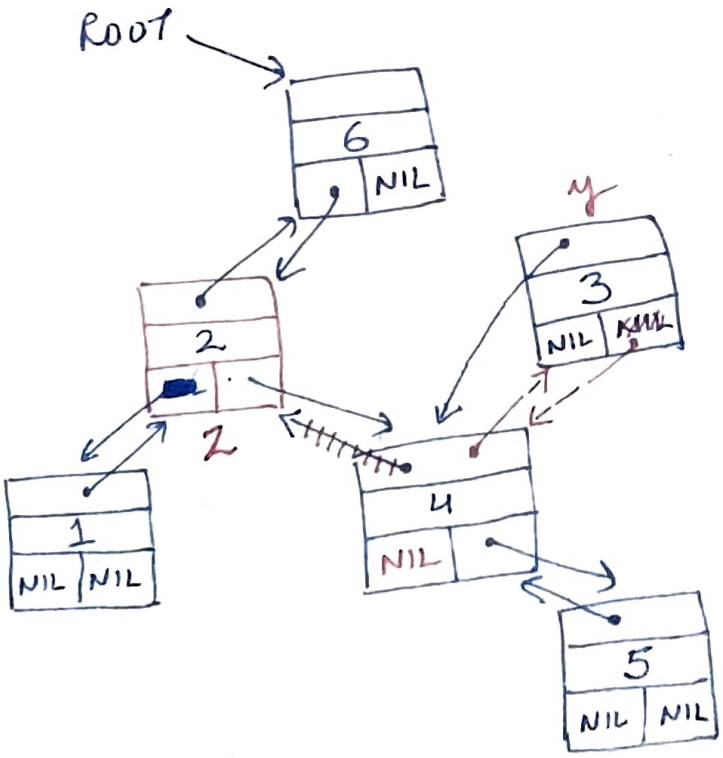


* TREE-DELETE(T, z)
(key value 2)

1. → z.left ≠ NIL
→ z.right ≠ NIL
→ y = TREE-MINIMUM(z.right)
[z.right has key value 4]

TREE-MINIMUM (z.right) returns node having key value 3.
So, y has the pointer pointing towards the node having key value 3.

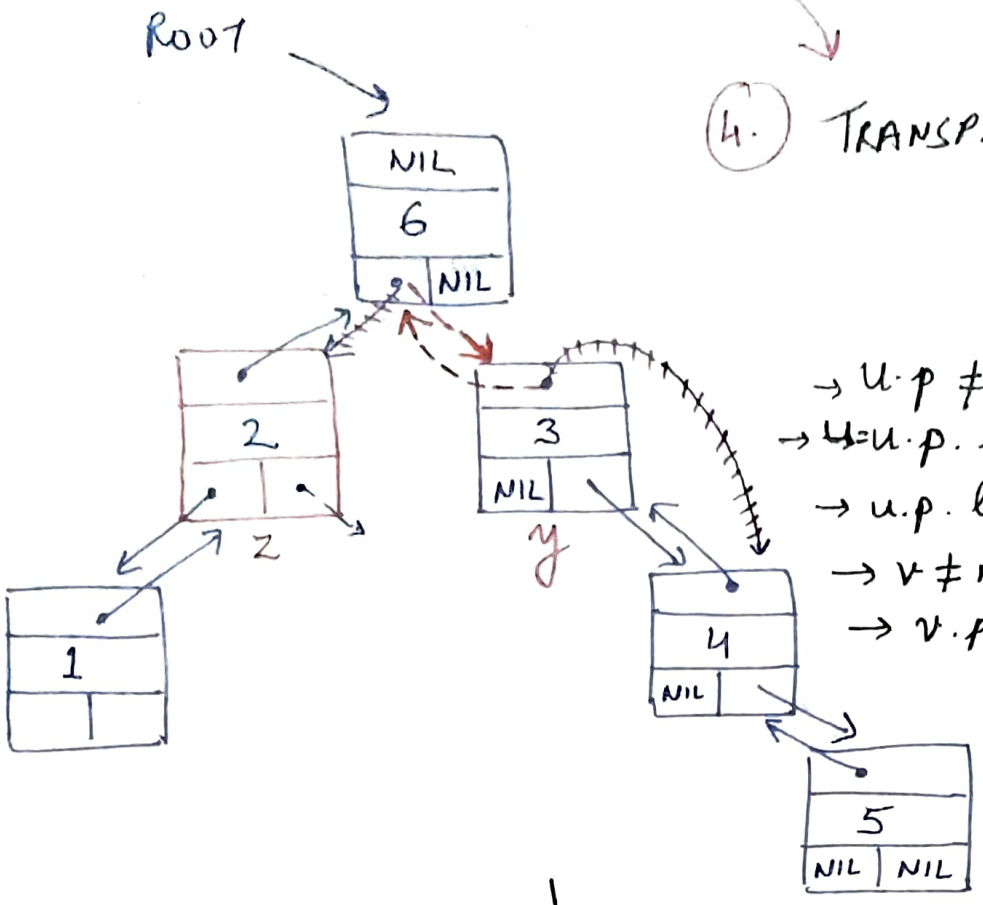
2. → y.p ≠ z
→ TRANSPLANT (T, y, y.right)
[TRANSPLANT (T, y, NIL)]



3. y.right = z.right
y.right.p = y

On Next Page

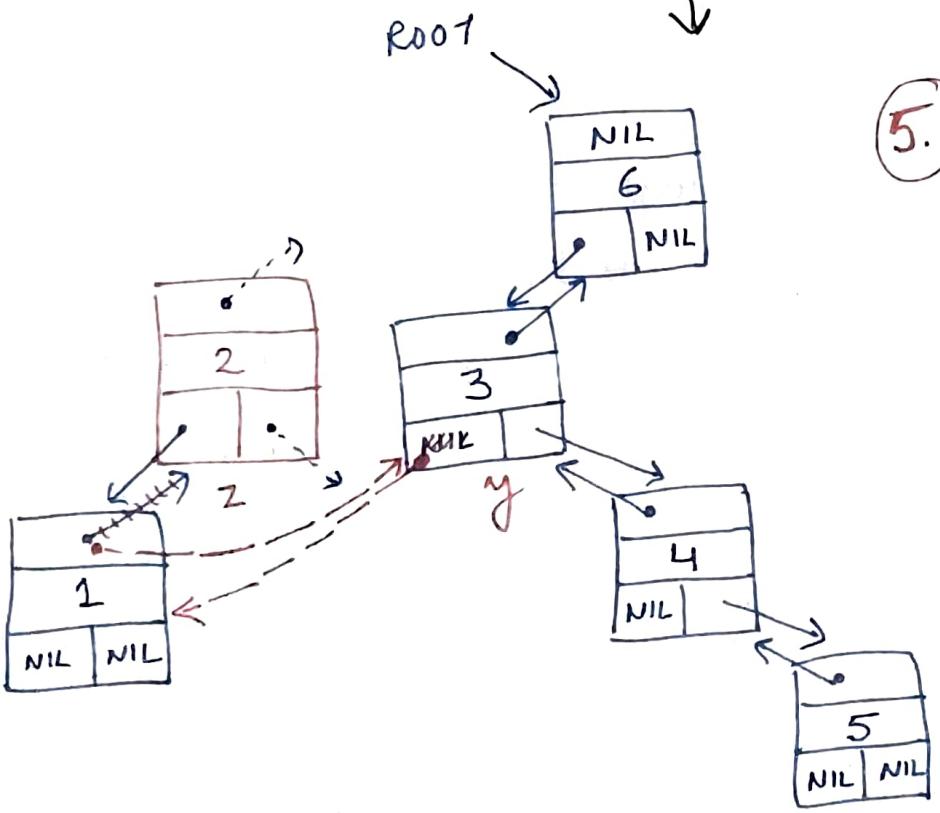
Contd.



4. TRANSPLANT(T, z, y)

(Node value 3)

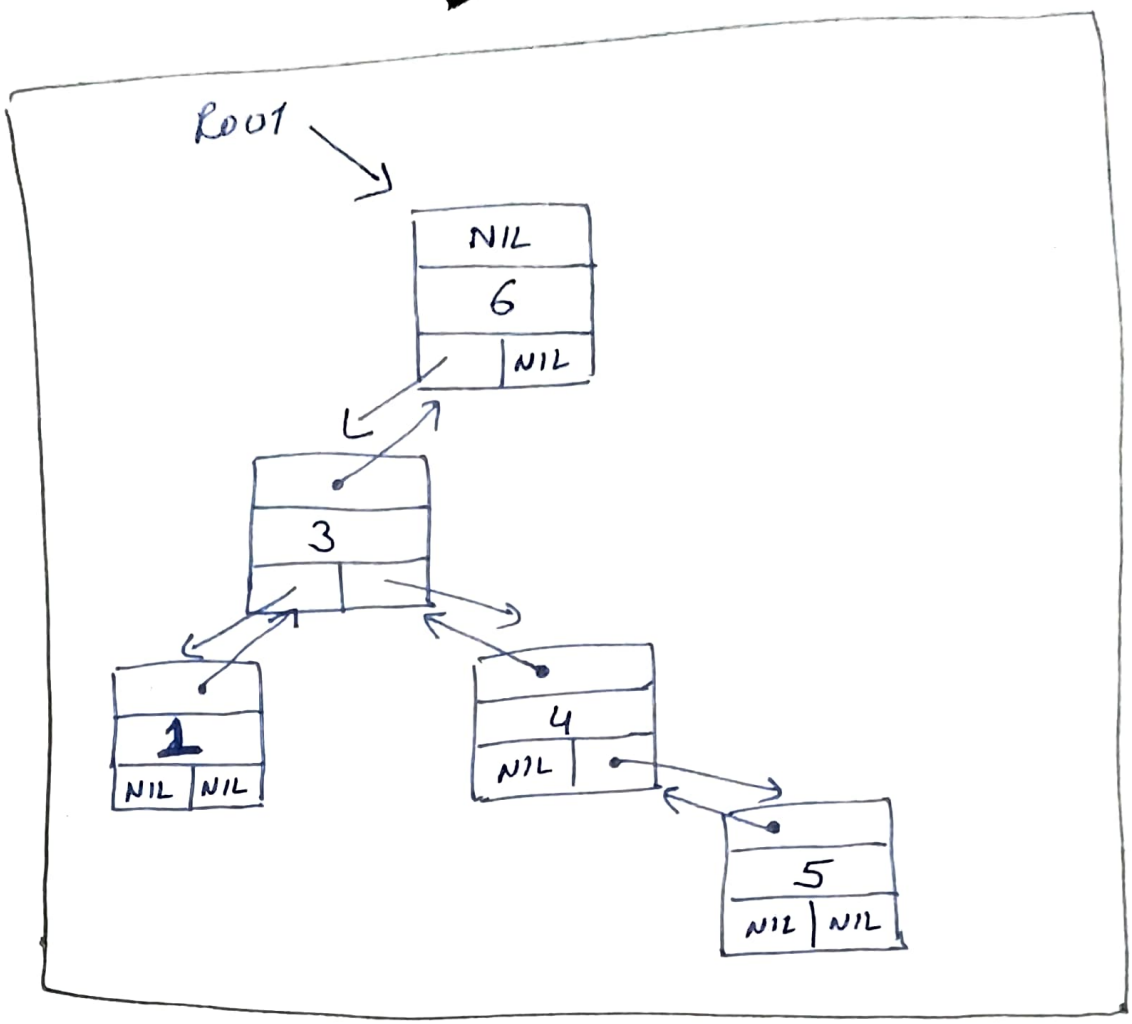
→ u.p ≠ NIL
→ u = u.p.left
→ u.p.left = v
→ v ≠ NIL
→ v.p = u.p



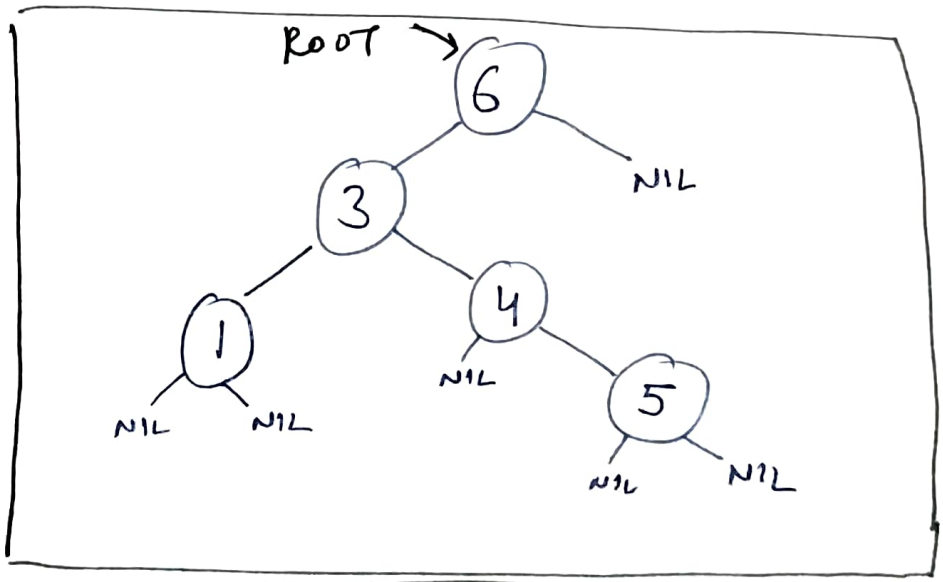
5. y.left = z.left
y.left.p = y

On Next Page

FINAL OUTCOME



OR



Final tree after deletion of node having key value 2.

NOTE:

* Remember to free the memory space occupied by [redacted] node [redacted] (z) after the function call for TREE-DELETE is over.