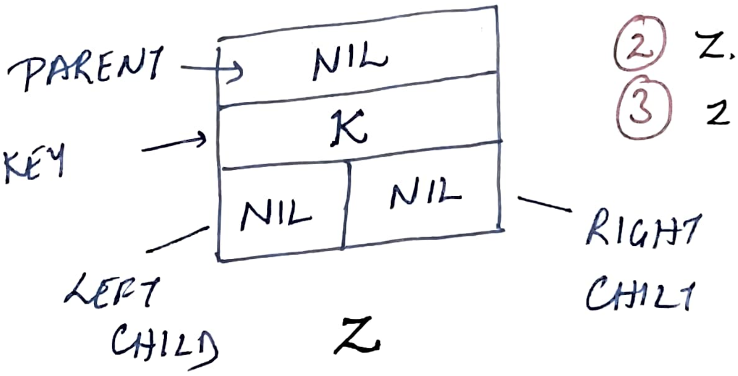
* BST INSERTION :

* For this algorithm, we assume that a new 'NODE z' needs to be inserted at correct position in the Tree.

* The tree can be empty or non-empty.

* So, for this we first create a new node z by dynamic memory allocation and initialize its values as follows :

① z.key = K
z.PARENT = NIL*
② z.LEFT = NIL
③ z.RIGHT = NIL



Here K is the value that is to be inserted.

NOTE

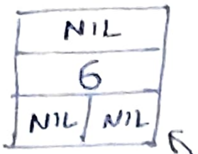
* In Cormen, As PARENT of z is updated and taken care off in algorithm itself, so this is not explitly initialized before the algo.

* INSERTION IN EMPTY TREE :

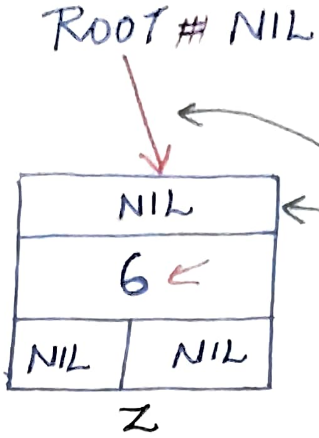
- INSERT(T, z)

T → TREE ROOT POINTER

z → NEW NODE



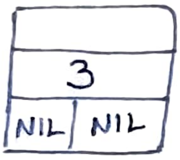
- Z NODE
- K=6



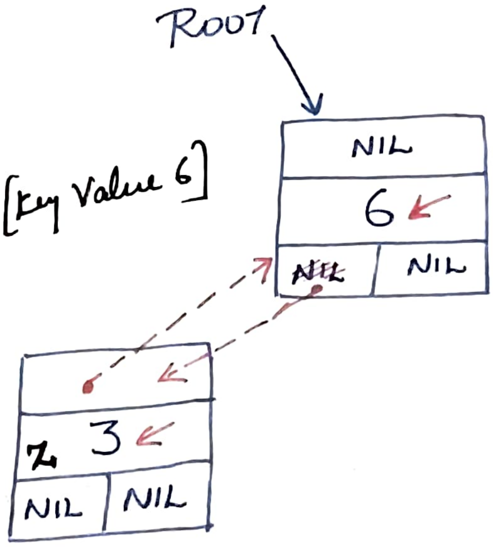
1. y = NIL
   x = T.root (ie. NIL)
   (so, x = NIL)
2. z.p = NIL [y]
3. y = NIL
   T.root = z

ANOTHER CASE :

- INSERT (T, z)



- Z NODE
- K = 3

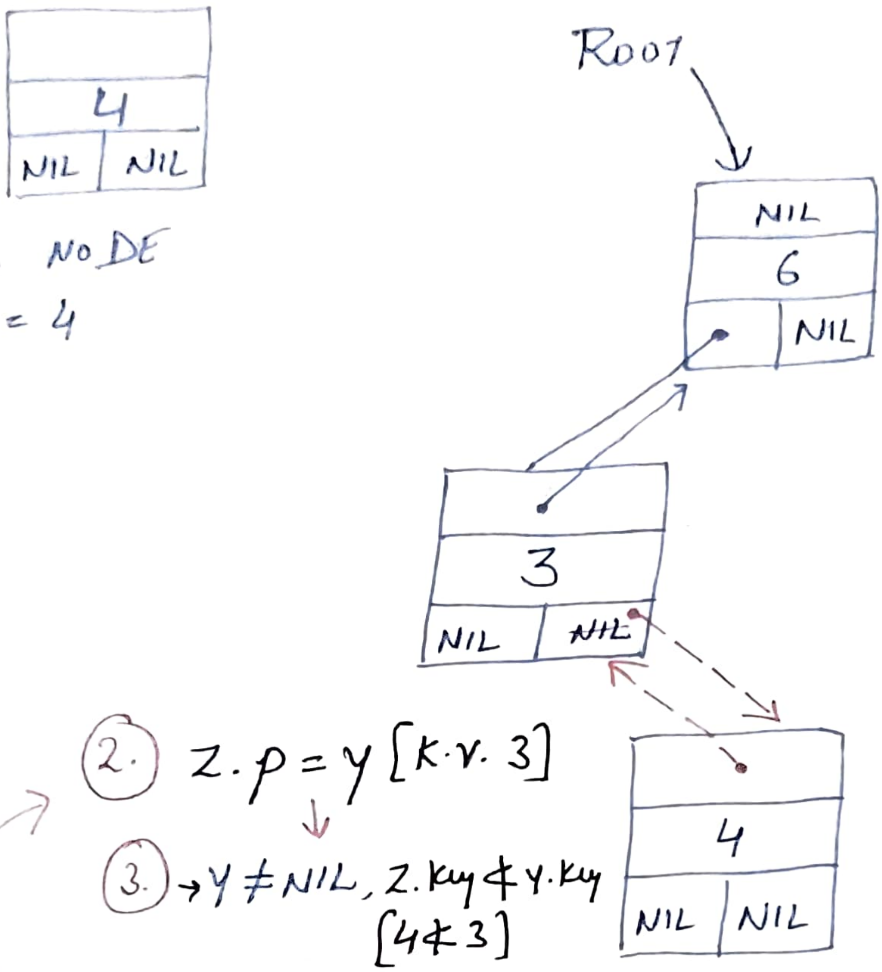


1. y = NIL
   x = T.root [key value 6]
   x ≠ NIL
   y = x [key value 6]
   3 < 6
   [z.key x.key]
   x = x.LEFT
   Now x = NIL [STOP]
2. z.p = y [key value 6]
3. y ≠ NIL,
   z.key < y.key
   - y.LEFT = z

* Thus, new nodes, 6 and then 3 are inserted sequentially in an initially empty BINARY SEARCH TREE.

ANOTHER CASE:

• INSERT(T, z)



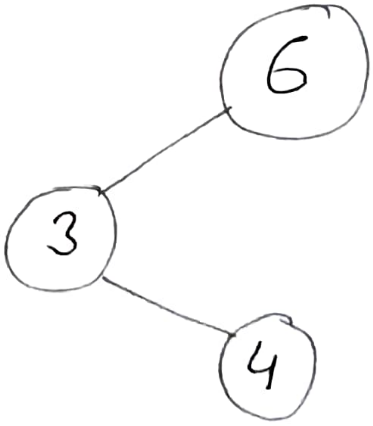
• z NODE
• k = 4

1. → y = NIL
→ x = T.root [KEY VALUE 6]
→ x ≠ NIL
→ y = x [K.V. 6]
→ z.key < x.key [4 < 6]
→ x = x.LEFT [K.V. 3]
→ x ≠ NIL
→ y = x [K.V. 3]
→ z.key ≮ x.key [4 ≮ 3]
→ x = x.RIGHT [NIL]
→ x = NIL [STOP]

2. z.p = y [K.V. 3]

3. → y ≠ NIL, z.key ≮ y.key [4 ≮ 3]
→ y.RIGHT = z

So, AFTER THIS, FINAL TREE IS:



And so on, you can further insert new nodes in This Tree (BST).