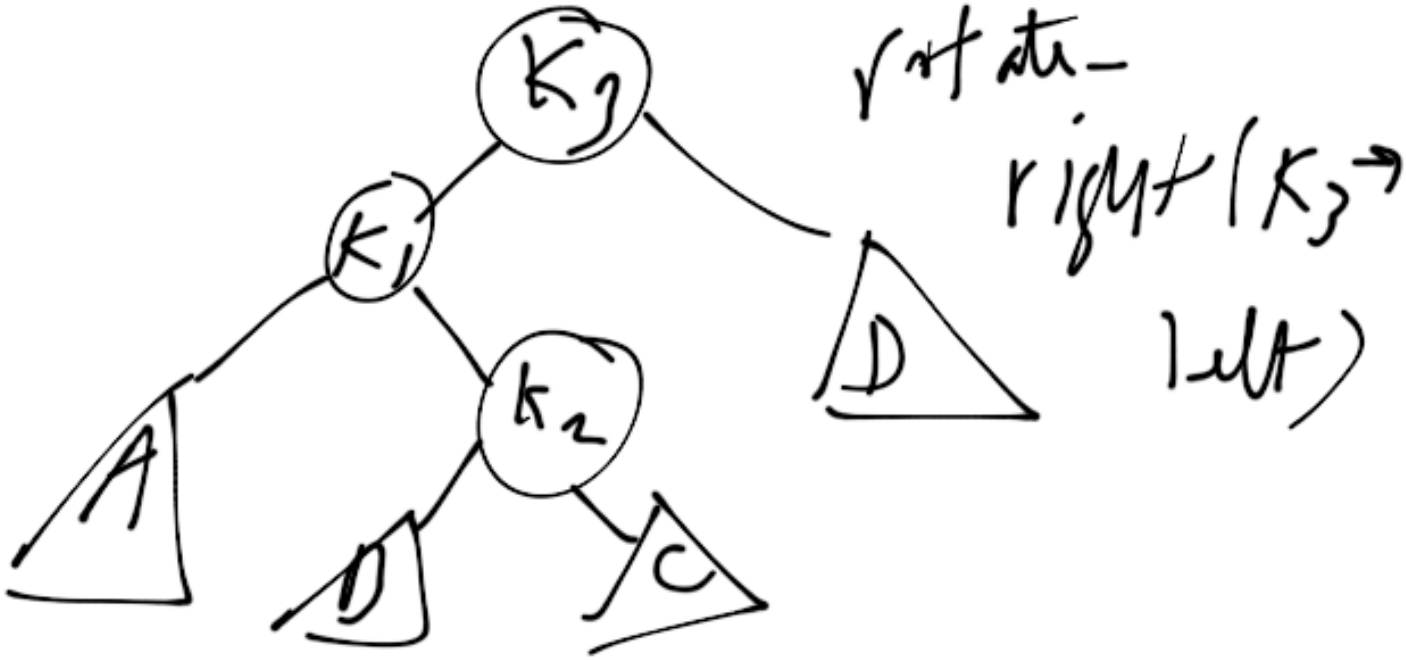
double-left rotation
In general (just the reverse of double-right rotation)

rotate_right

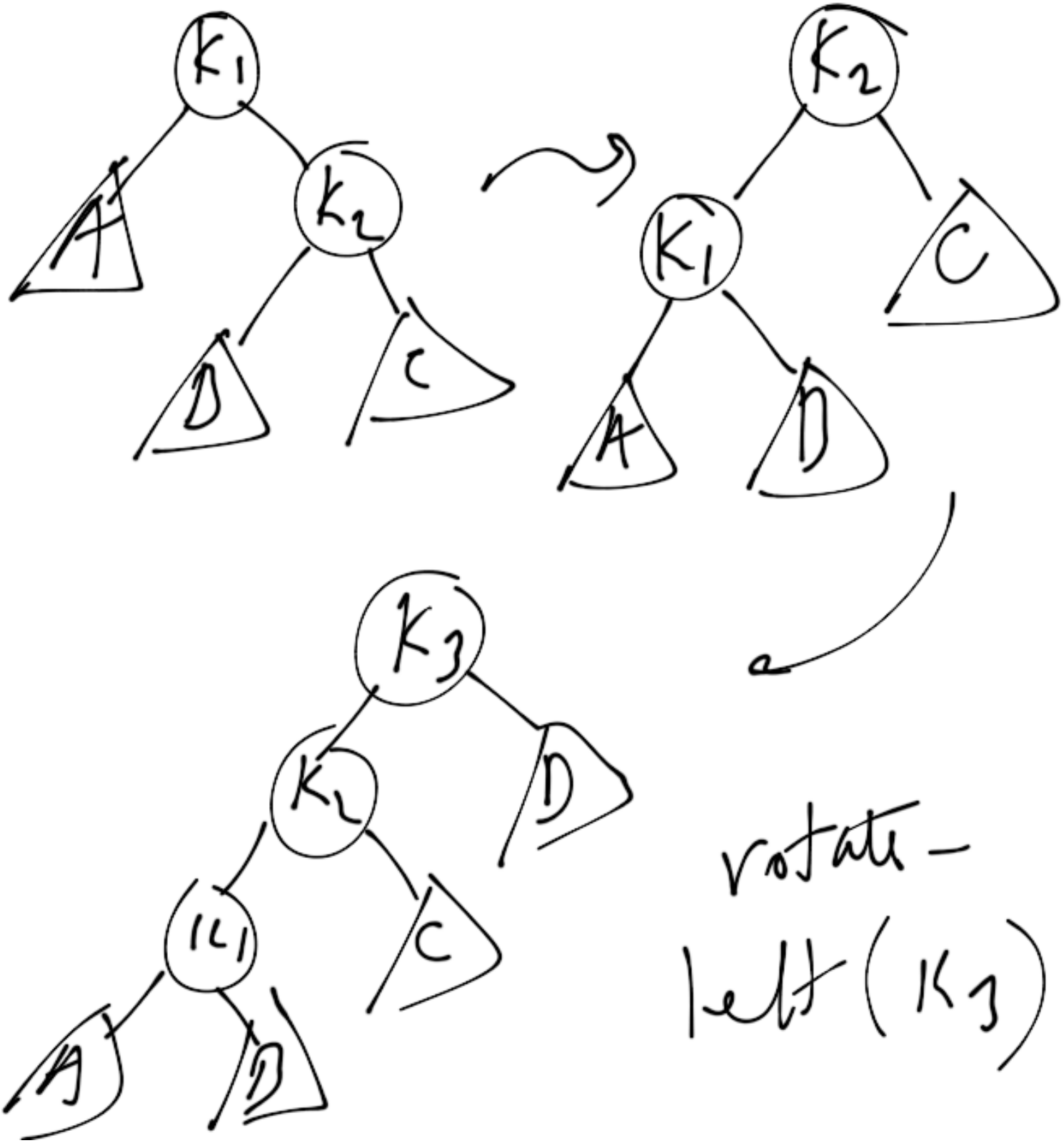
followed by

rotate_left



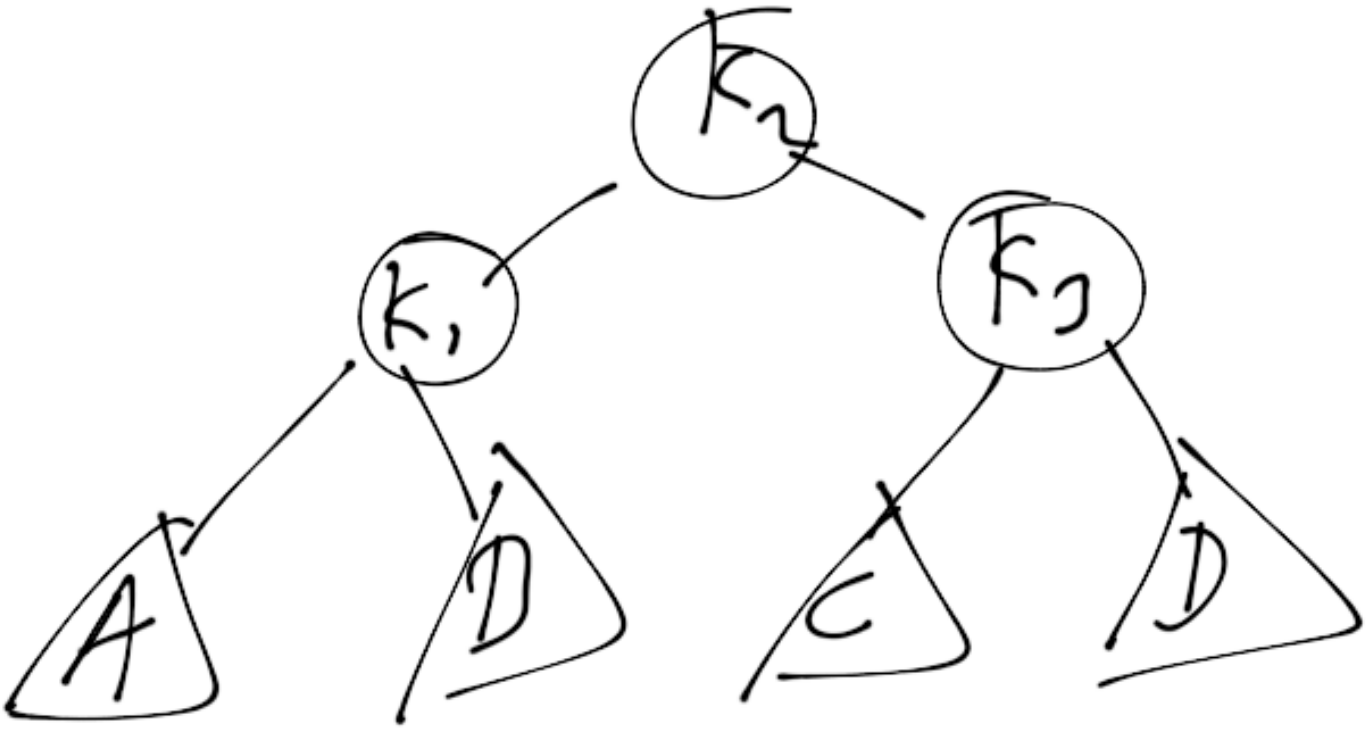
rotate-right(K3→left)

rotate_right($k_3 \rightarrow$ left)

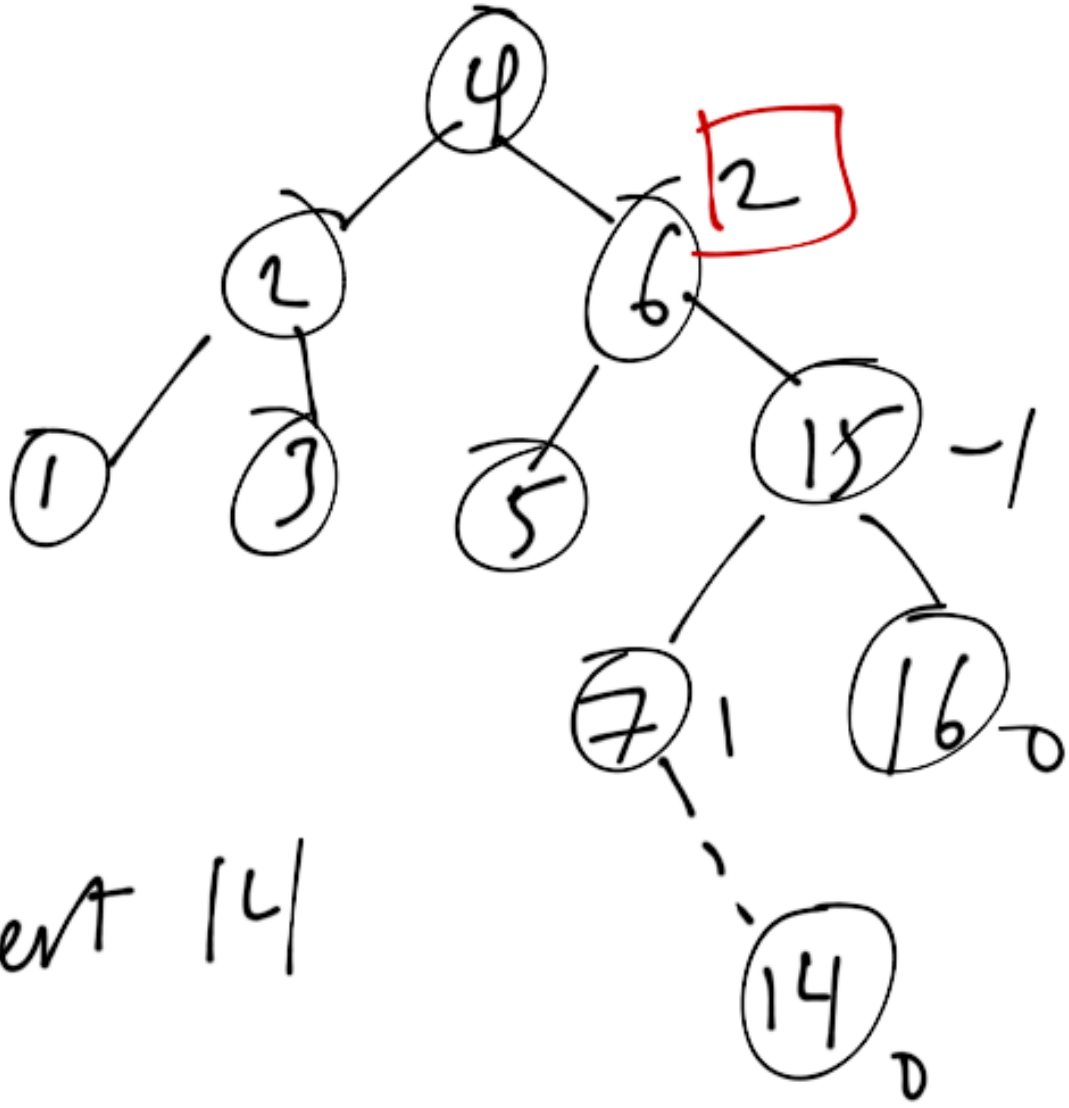


rotate-left(k_3)

rotate_left (k_3)



our tree is now

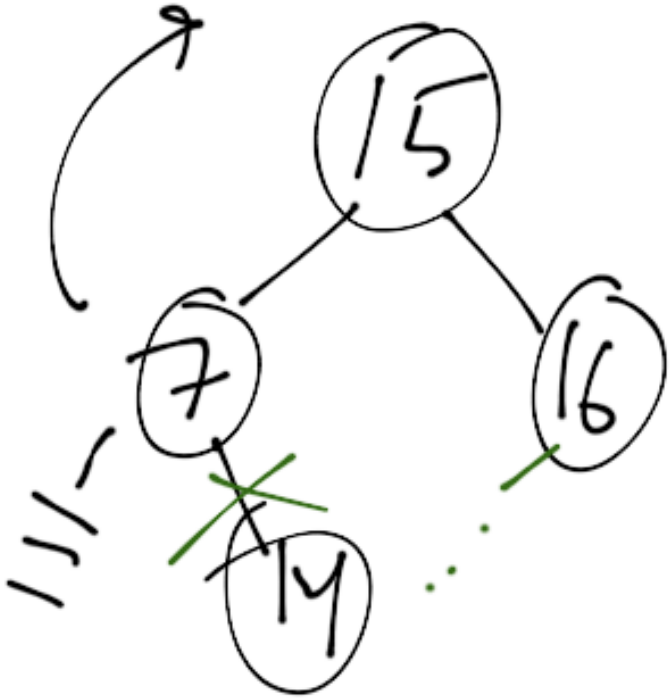


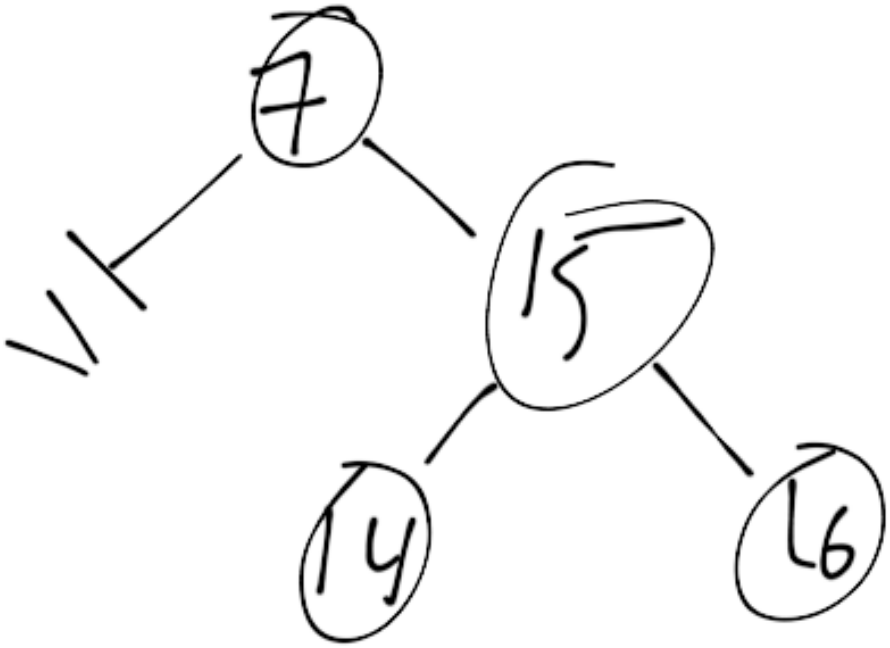
insert 14

rotate_left (right child)

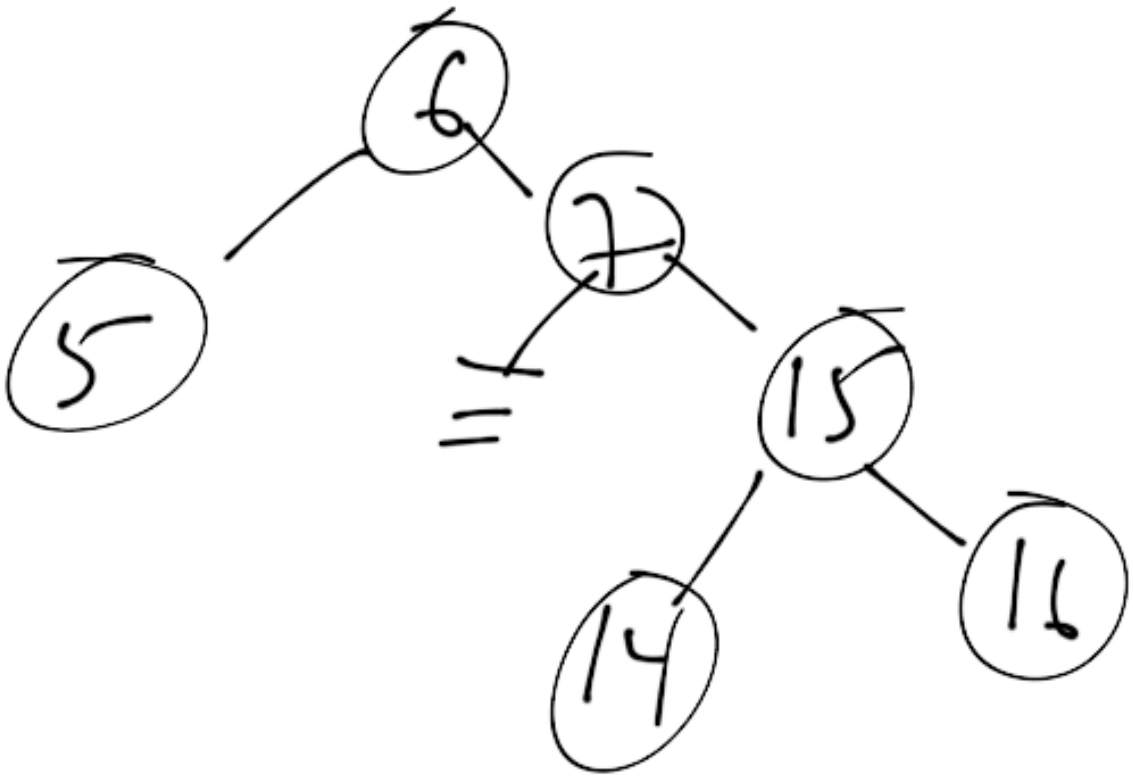
then rotate_right (self)

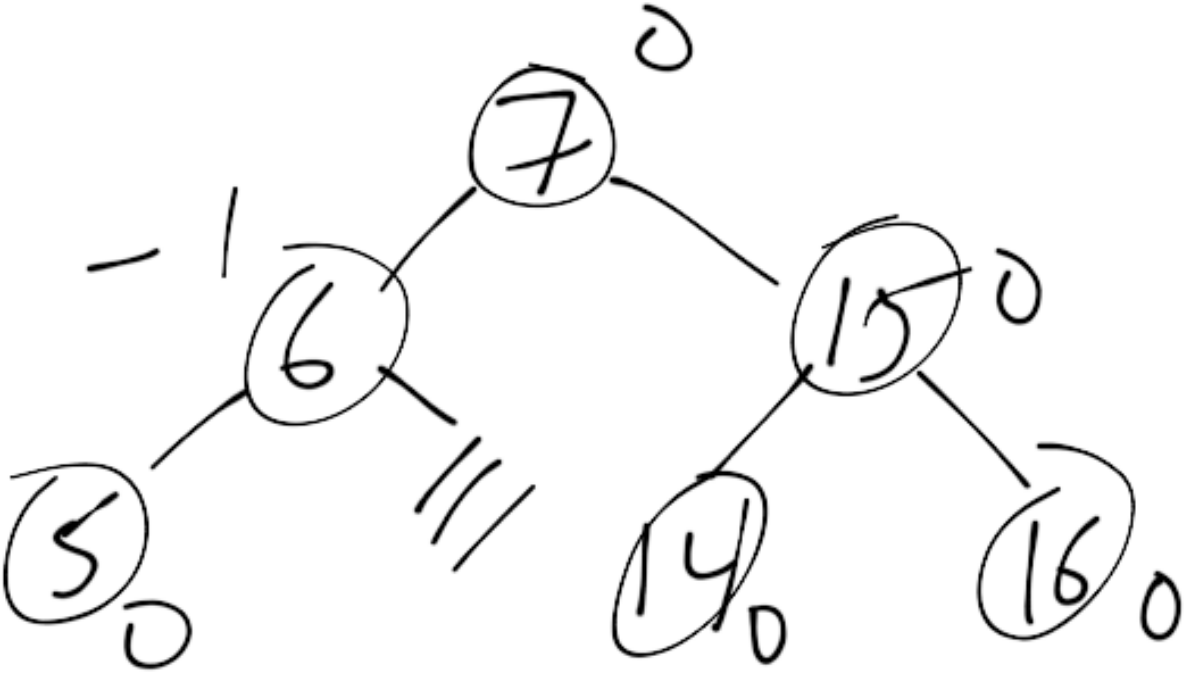
rotate_left(15)



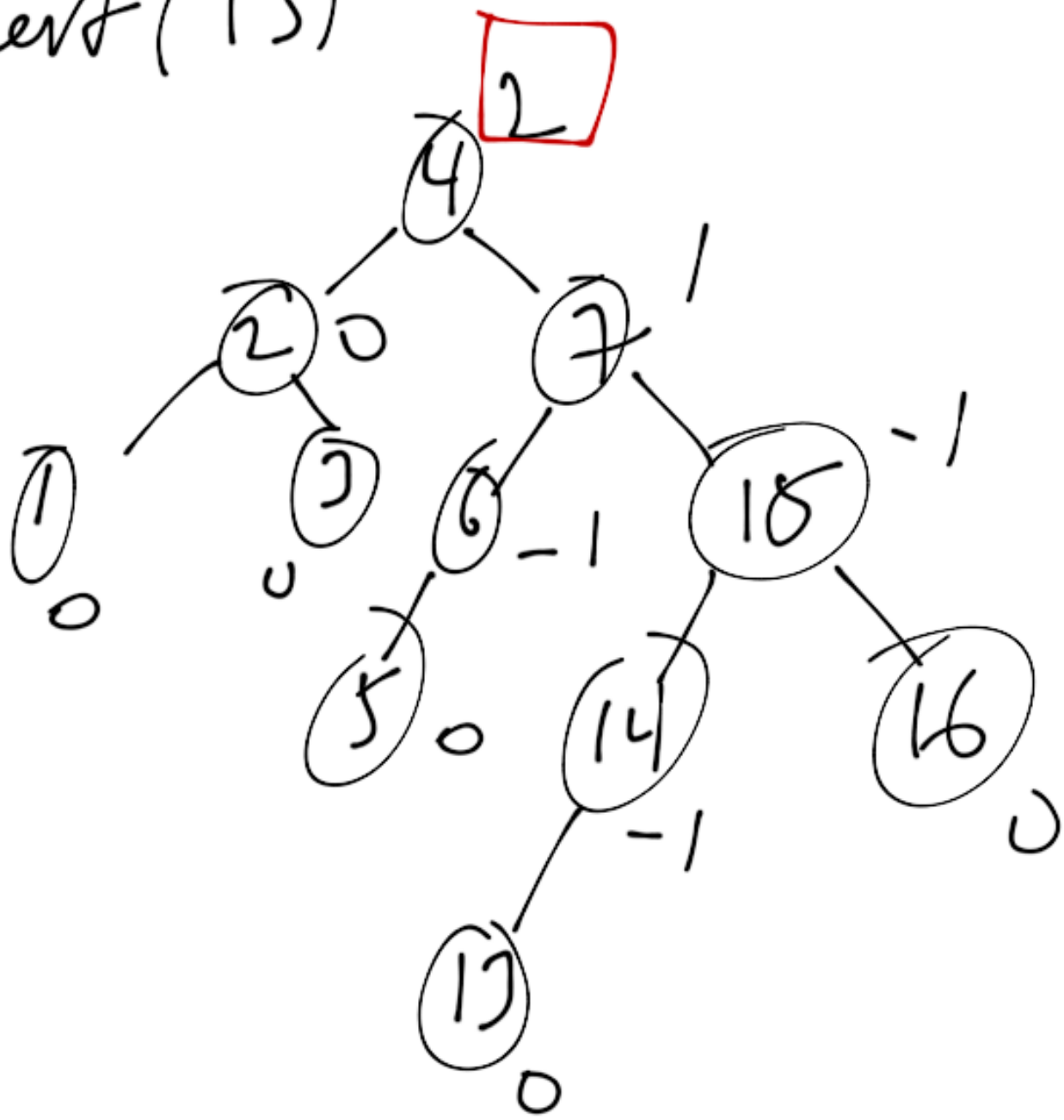


rotate_right(6)

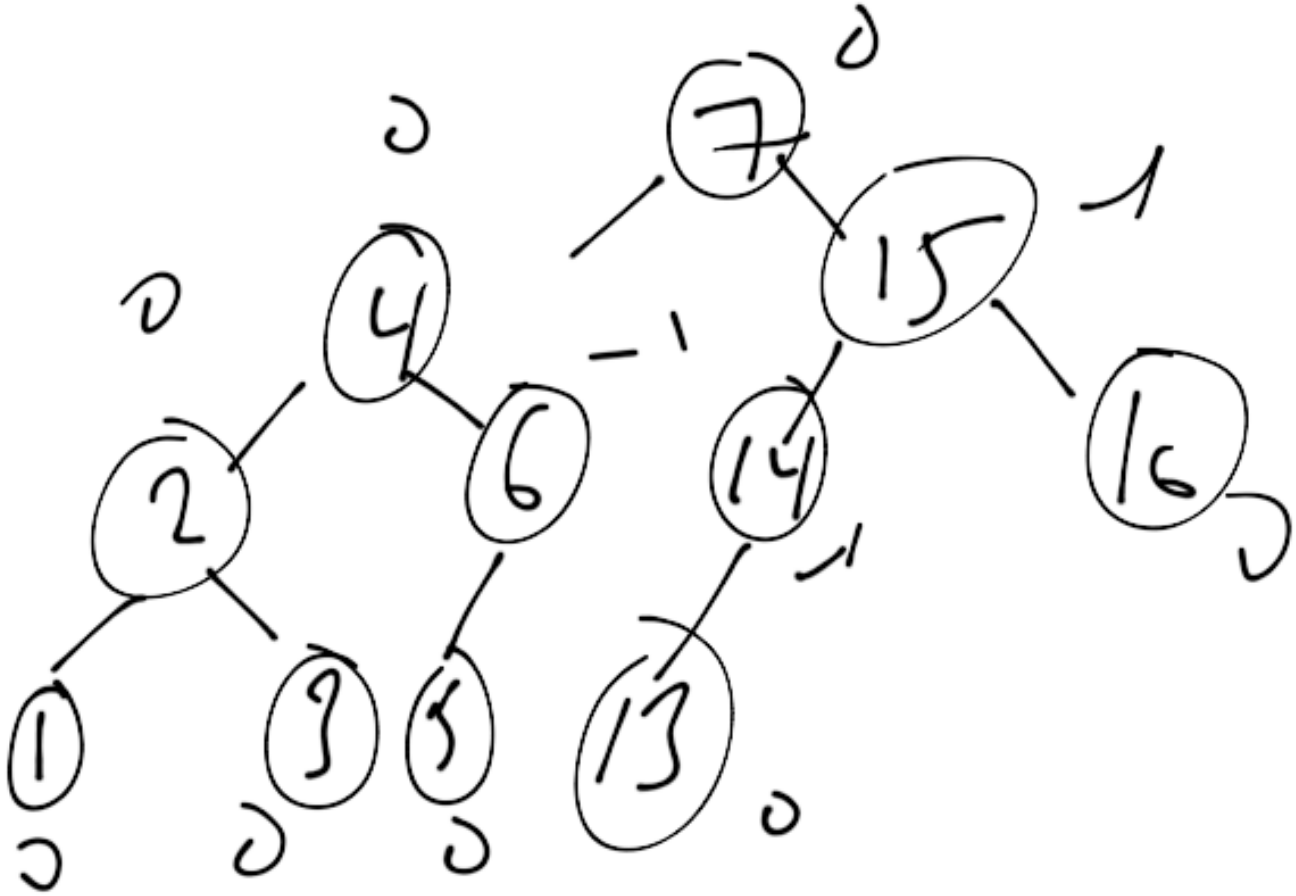




Insert (13)



single rotation



to implement our BST node_t just needs an additional int bal; set to 0 initially

```
insert(const T &x,
       node_t *&t)
    int dh = 0; (delta
                 height)
if (t == NULL) {
    t = new node_t(
        x, NULL, NULL);
    dh = 1
}
   :
// return(dh) at end...
```

```
else if (x < t->data) {
  // go down left subtree
  if (insert(x, t->left)) {
    t->bal--; // left subtree
              // increased
    if (t->bal == -1)
      deltah = 1;
    else if (t->bal == -2) {
      if (t->left->bal == 1)
        rotate_right(t->left)
```

left child ← "right heavy"

```
      rotate_left(t)
```

3 3

```
else if (x > t->data)
    if (insert(x, t->right)) {
        t->bal++;
        if (t->bal == 1)
            sh = 1;
        else if (t->bal == 2)
            if (t->right->bal == -1)
                rotate_left(t->right)
            rotate_right(t)
    }
} else; // duplicate (x == t->data)
return sh;
```

right child

left heavy

What about delete?

Alg.:

```
if (t == NULL)
    return ∅
```

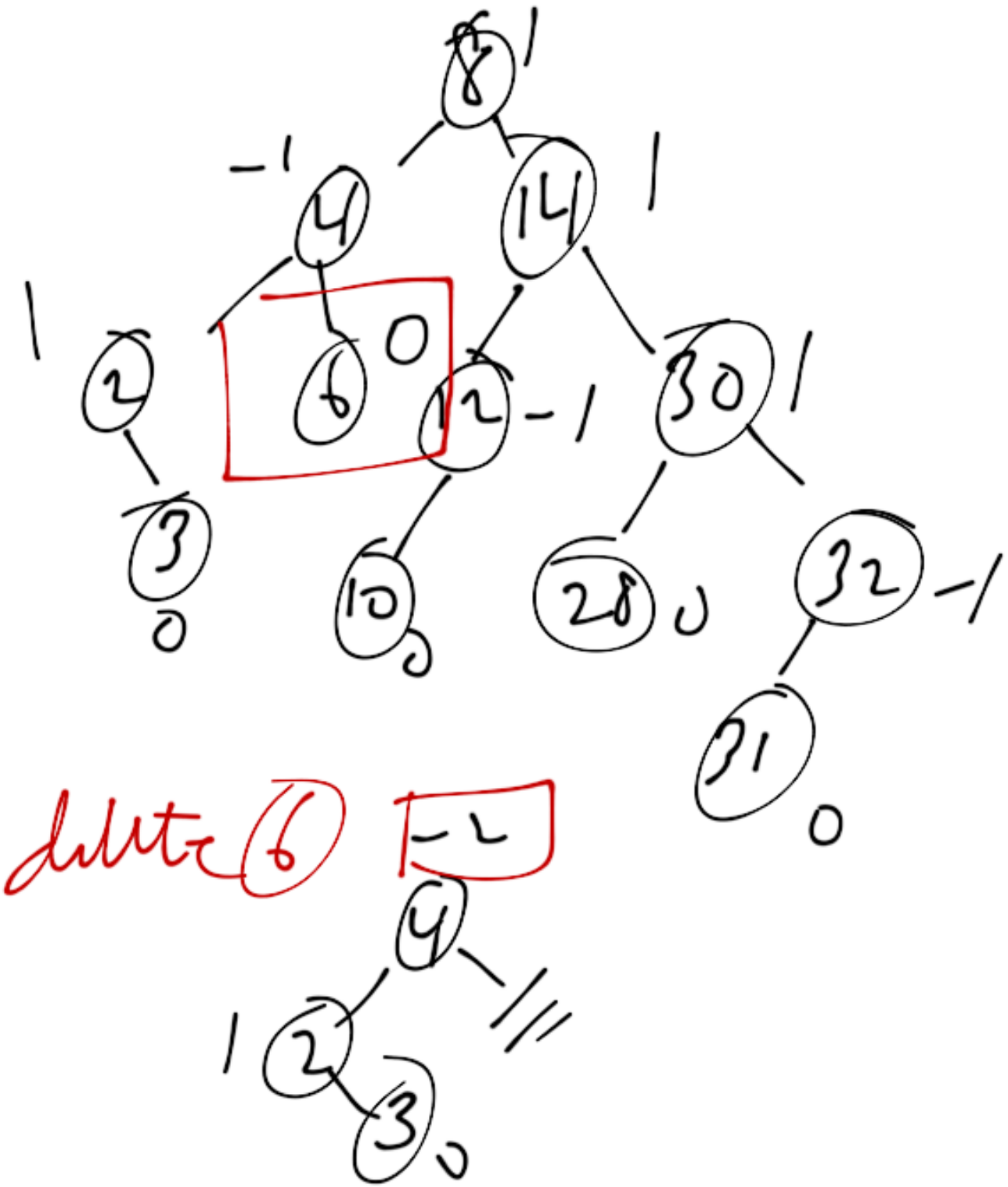
```
if (x < t->data) {
    if (erase(x, t->left)) {
        // almost like
        insert(x, t->right)
        t->bal++; // height of
                  left store
        if (t->bal == 0) decreased
            dh = 1
        else if (t->bal == 2)
            if (t->right->bal == -1)
                rotate_left(t->right)
            rotate_right(t)
    } if (t->bal == 0) dh = 1
```

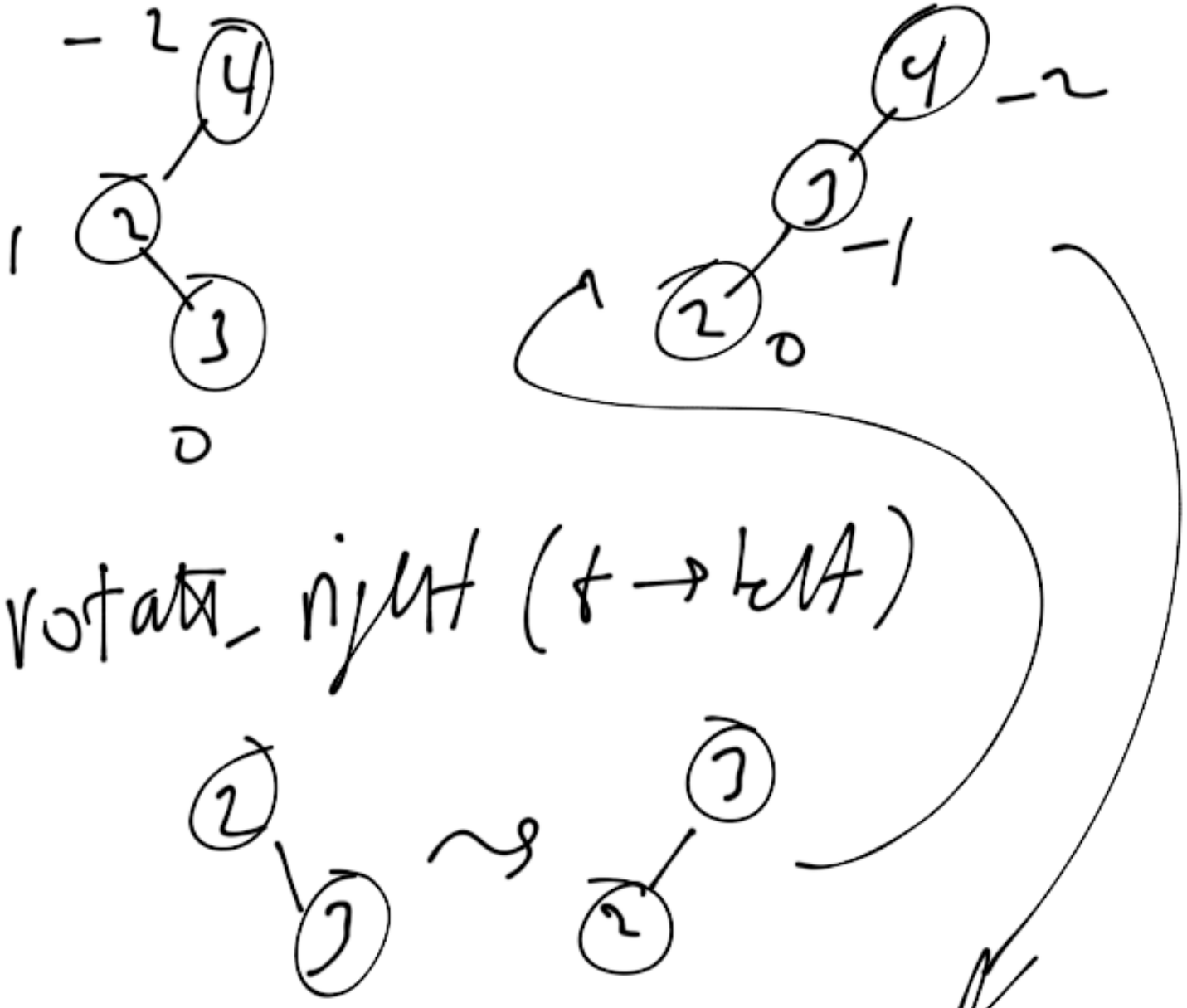
```
else if (x > t->data)
    if (erase(x, t->right)) {
        // almost like
        insert(x, t->left)
        .
        .
        .
        .
        .
        .
        .
        .
}
```

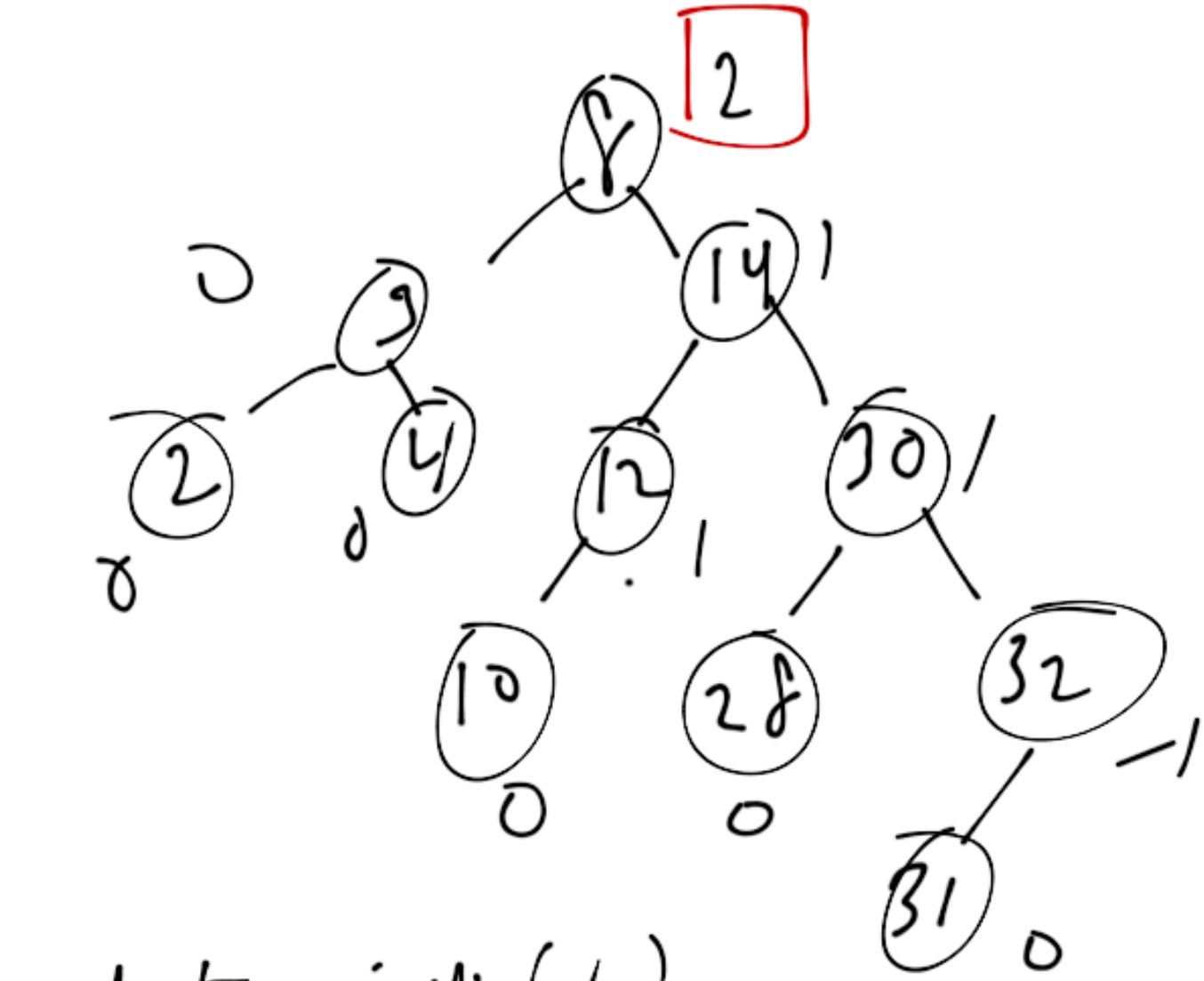
```
else if (t->left != NULL &&
         t->right != NULL)
// x == t->data;
// want to delete t, t has 2 children
  t->bal--;
// almost like insert(x, t->left);
if (t->bal == 0) oh = 1
else if (t->bal == -2)
    if (t->left->bal == 1)
        rotate_right(t->left)
    rotate_left(t)
} if (t->bal == 0) oh = 1
```

```
else { // one subtree NULL
    // set current node t
    // to non-NULL subtree
    node_t *old = t;
    t = (t->left != NULL) ?
            t->left : t->right;
    delete old;
        // calls node_t
        // destructor
    sh = 1
}
return sh
```

Insert 6, 4, 2, 8, 10, 12,
14, 32, 30, 28, 3, 31




rotate_right(t->left)
rotate_left(t)



Rotate-right(+)