If current node contains the point to be deleted

1. If node to be deleted is a leaf node, simply delete it (Same as BST Delete)
2. If node to be deleted has right child as not NULL (Different from BST)
    1. Find minimum of current node's dimension in right subtree.
    2. Replace the node with above found minimum and recursively delete minimum in right subtree.
3. Else If node to be deleted has left child as not NULL (Different from BST)
    1. Find minimum of current node's dimension in left subtree.
    2. Replace the node with above found minimum and recursively delete minimum in left subtree.
    3. Make new left subtree as right child of current node.

2) If current doesn't contain the point to be deleted

1. If node to be deleted is smaller than current node on current dimension, recur for left subtree.
2. Else recur for right subtree.

Why 1.b and 1.c are different from BST?

In BST delete, if a node's left child is empty and right is not empty, we replace the node with right child. In K D Tree, doing this would violate the KD tree property as dimension of right child of node is different from node's dimension. For example, if node divides point by x axis values. then its children divide by y axis, so we can't simply replace node with right child. Same is true for the case when right child is not empty and left child is empty.

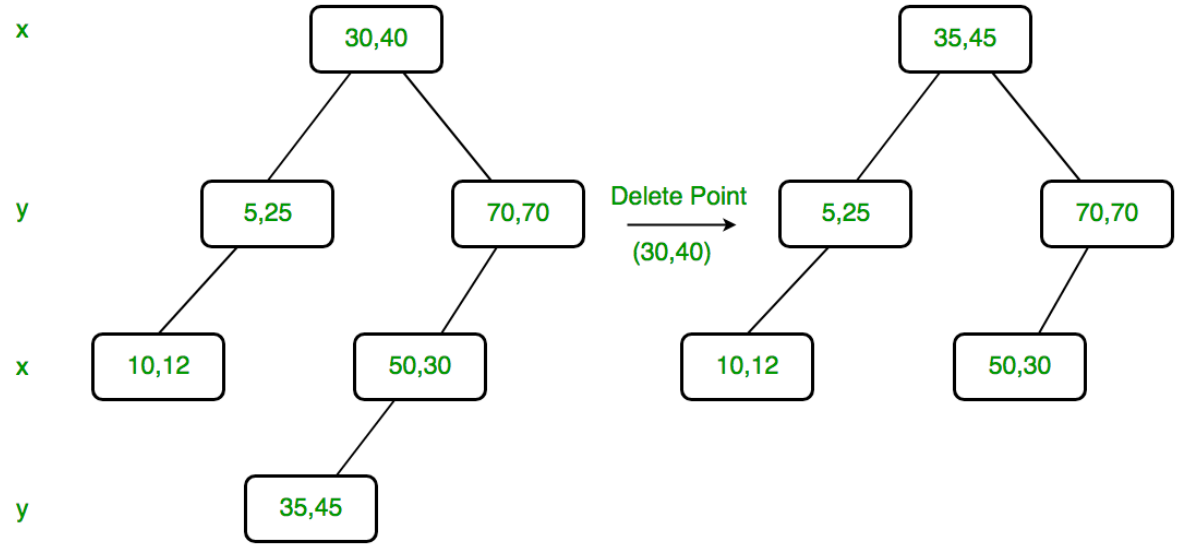
Why 1.c doesn't find max in left subtree and recur for max like 1.b?
Doing this violates the property that all equal values are in right subtree. For example, if we delete (!0, 10) in below subtree and replace if with

```
Wrong Way (Equal key in left subtree after deletion)
            (5, 6)                                (4, 10)
             /              Delete(5, 6)           /
        (4, 10)            ------------>      (4, 20)
             \
           (4, 20)
```

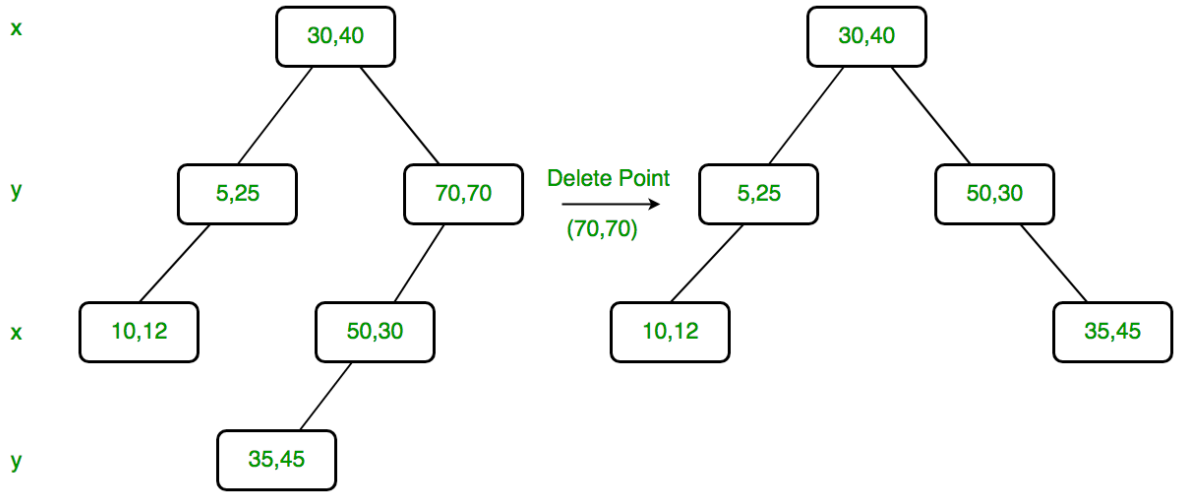
```
Right way (Equal key in right subtree after deletion)
             (5, 6)                          (4, 10)
             /              Delete(5, 6)           \
        (4, 10)             ------------>       (4, 20)
             \
             (4, 20)
```

Example of Delete:

Delete (30, 40): Since right child is not NULL and dimension of node is x, we find the node with minimum x value in right child. The node is (35, 45), we replace (30, 40) with (35, 45) and delete (30, 40).



Delete (70, 70): Dimension of node is y. Since right child is NULL, we find the node with minimum y value in left child. The node is (50, 30), we replace (70, 70) with (50, 30) and recursively delete (50, 30) in left subtree. Finally we make the modified left subtree as right subtree of (50, 30).


x
y
x
y
30,40
5,25
70,70
10,12
50,30
35,45
Delete Point
(70,70)
30,40
5,25
50,30
10,12
35,45