KD-Tree

K Dimensional Tree

- Árbol Binario de Búsqueda donde cada nodo es un punto en K dimension
- Un nodo (no hoja) divide el espacio en dos partes medio-espacio (half-space)
- Puntos a la izquierda de este espacio son representados por el subtree izquierdo, los puntos a la derecha por el subtree de la derecha.
- Veremos un ejemplo en 2D
- El root es alineado con el eje x, el hijo del root con el eje y, el nieto con el eje x, y así sucesivamente.

Generalization

- Enumeramos los planos
  - 0,1,2,3… K-1
  - Un punto a una profundidad D será alineada con el plano A
  - A = D % K

Determinar en qué subtree caerá un punto

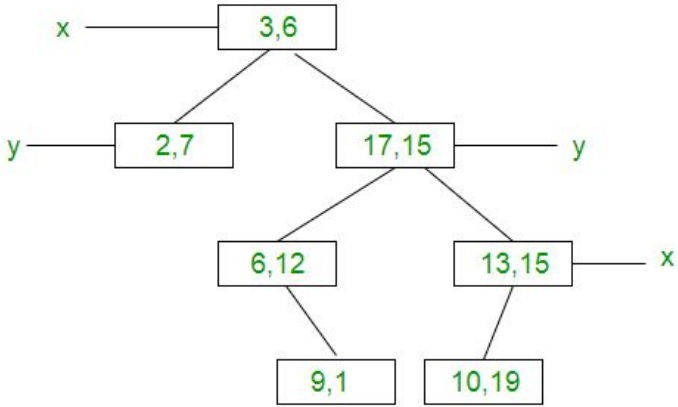
Si el root es alineado con plano A, entonces el subtree de la izquierda contendrá los puntos inferiores al plano A y el subtree de la derecha los puntos superiores o iguales a plano A

Creación del KD-Tree

Consider following points in a 2-D plane:

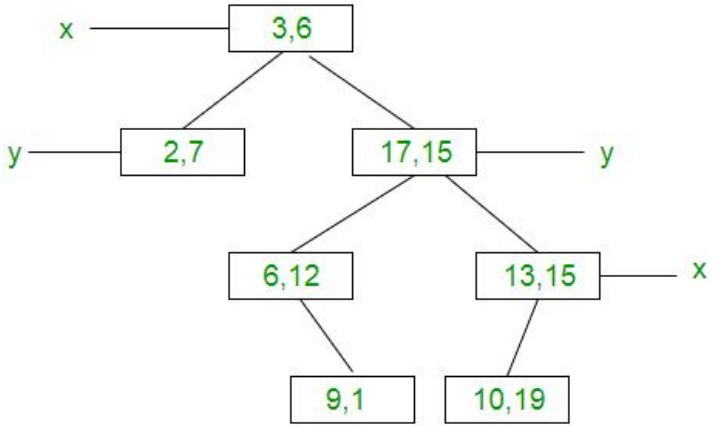
(3, 6), (17, 15), (13, 15), (6, 12), (9, 1), (2, 7), (10, 19)

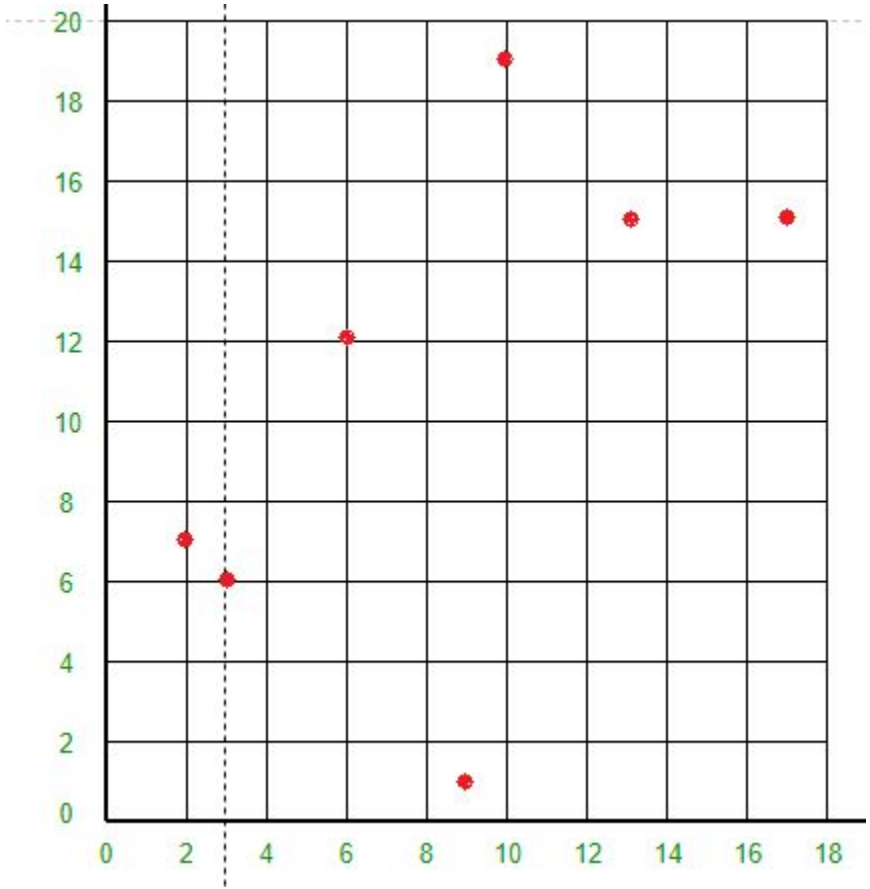
1. Insert (3, 6): Since tree is empty, make it the root node.
2. Insert (17, 15): Compare it with root node point. Since root node is X-aligned, the X-coordinate value will be compared to determine if it lies in the rightsubtree or in the right subtree. This point will be Y-aligned.
3. Insert (13, 15): X-value of this point is greater than X-value of point in root node. So, this will lie in the right subtree of (3, 6). Again Compare Y-value of this point with the Y-value of point (17, 15) (Why?). Since, they are equal, this point will lie in the right subtree of (17, 15). This point will be X-aligned.
4. Insert (6, 12): X-value of this point is greater than X-value of point in root node. So, this will lie in the right subtree of (3, 6). Again Compare Y-value of this point with the Y-value of point (17, 15) (Why?). Since, 12 < 15, this point will lie in the left subtree of (17, 15). This point will be X-aligned.
5. Insert (9, 1):Similarly, this point will lie in the right of (6, 12).
6. Insert (2, 7):Similarly, this point will lie in the left of (3, 6).
7. Insert (10, 19): Similarly, this point will lie in the left of (13, 15).a



Creacion

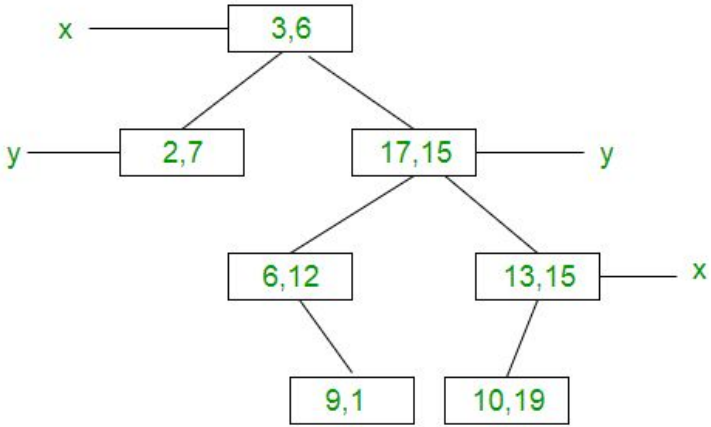
Insert (3, 6)

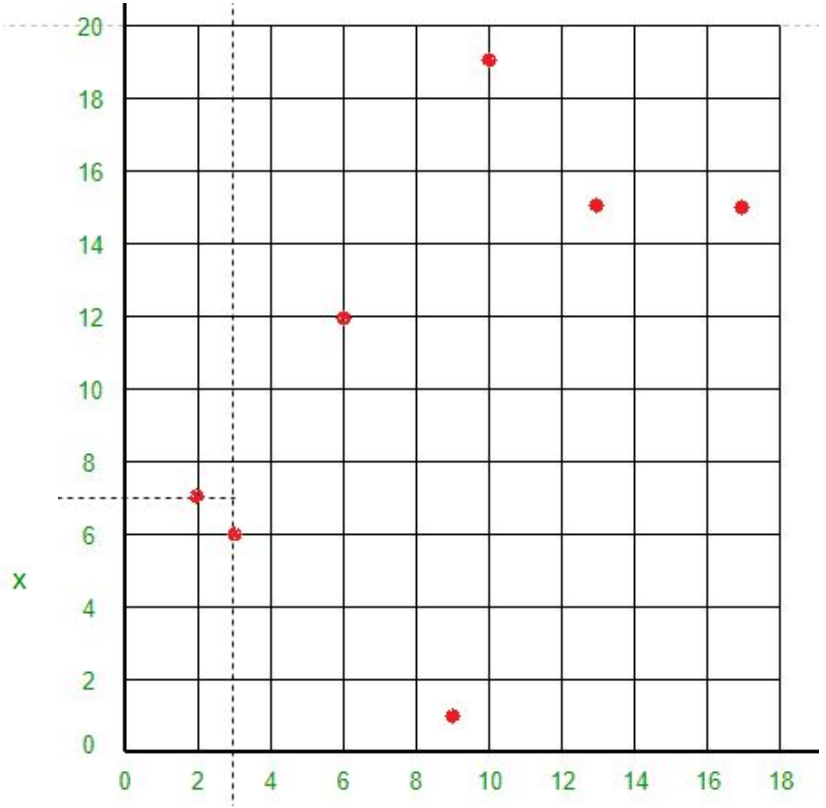




Creacion

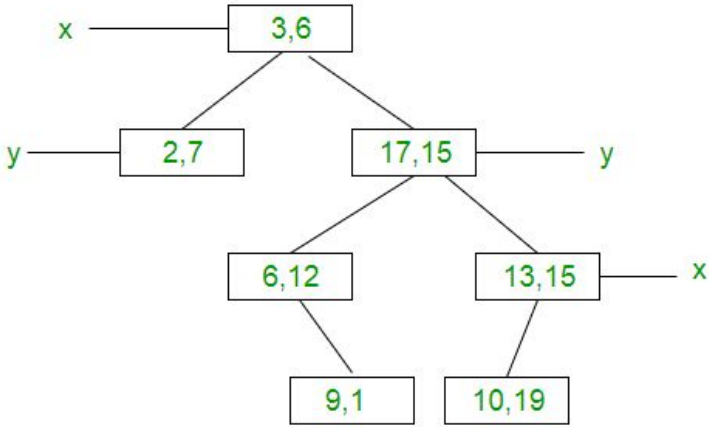
Insert (2, 7)

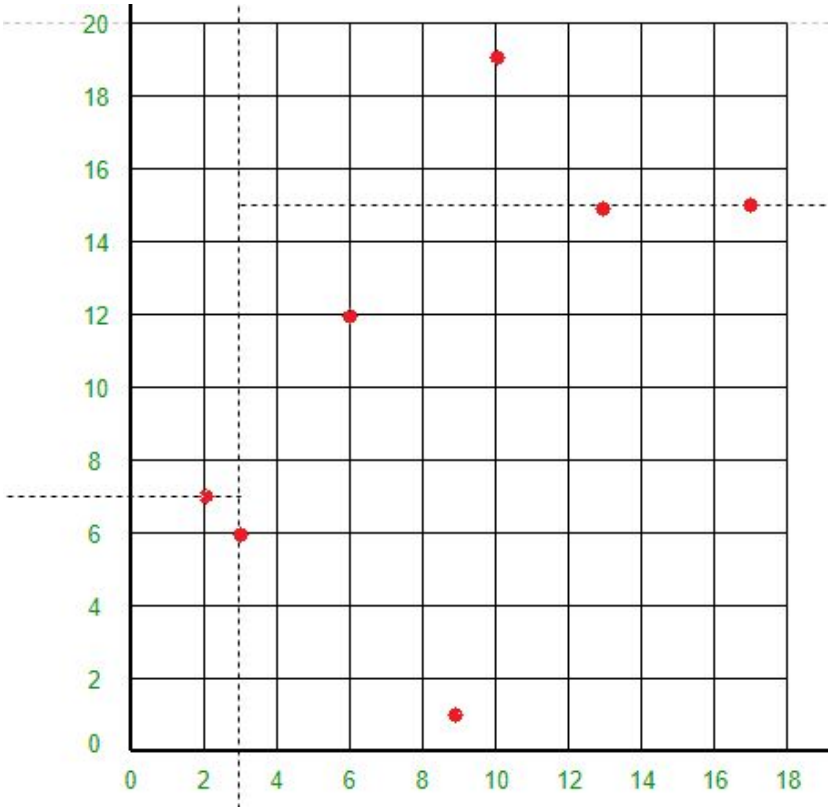




Creacion

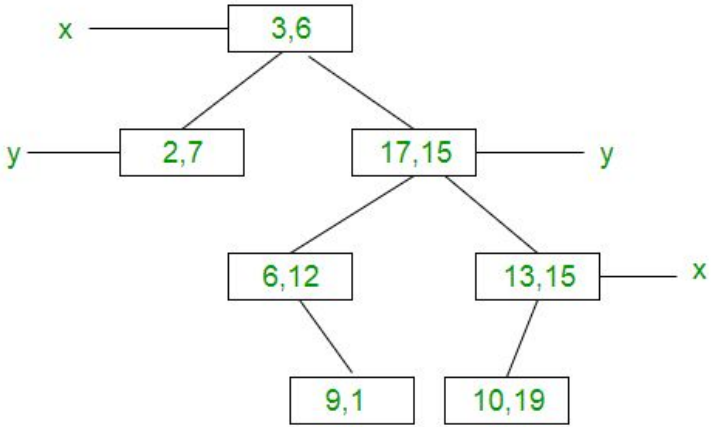
Insert (17, 15)

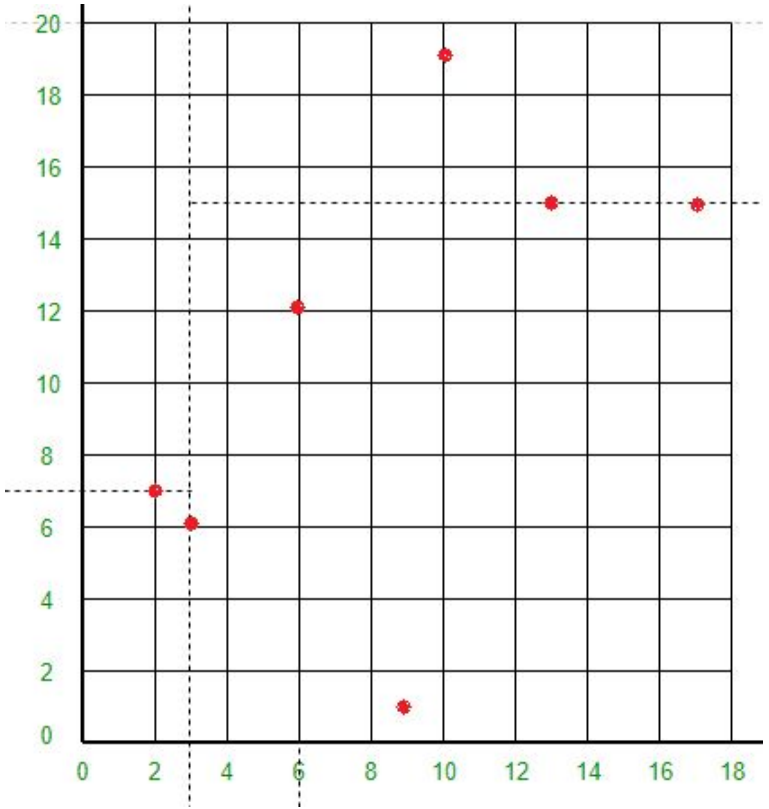




Creacion

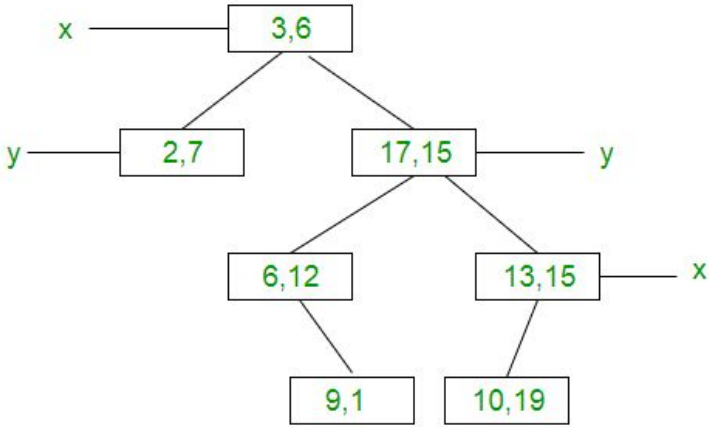
Insert (6, 12)

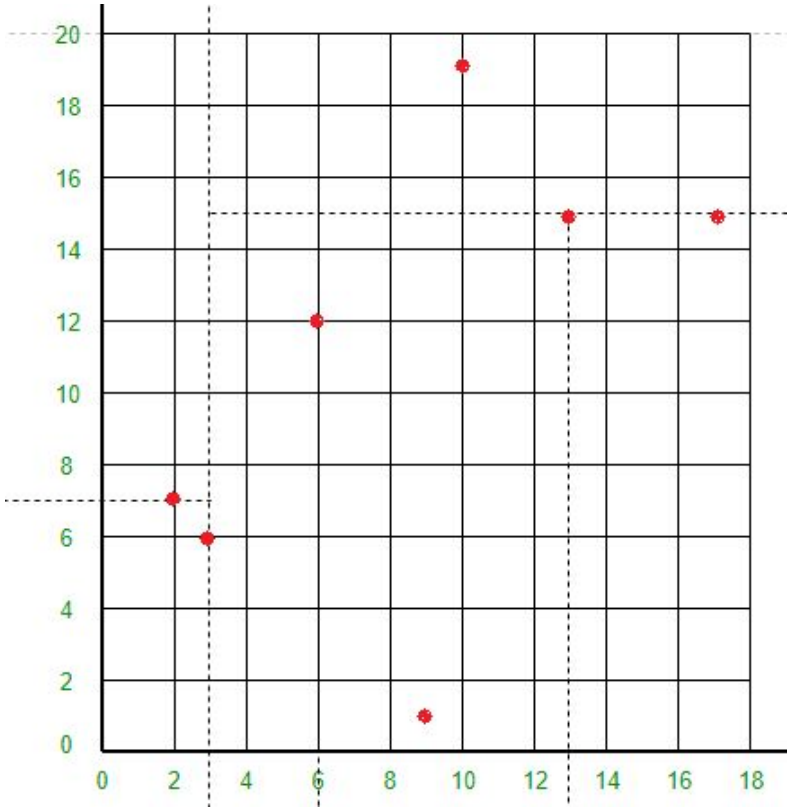




Creacion

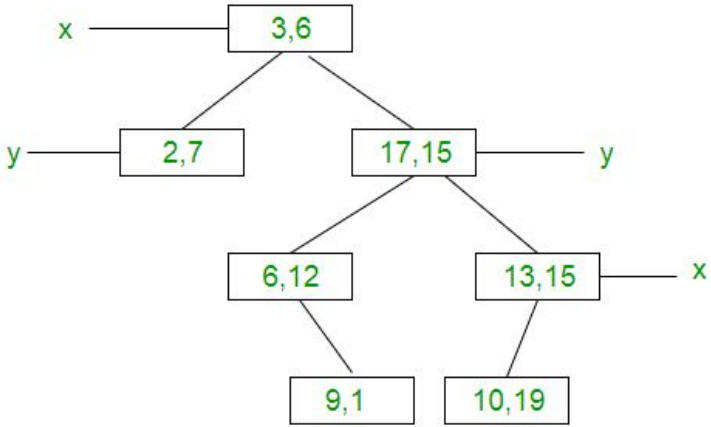
Insert (13, 15)

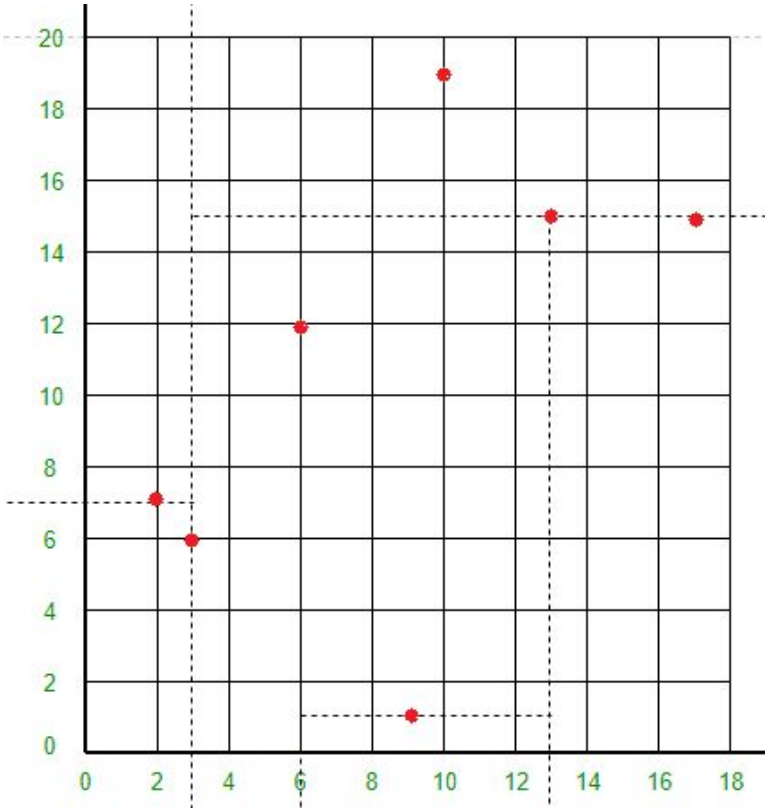




Creacion

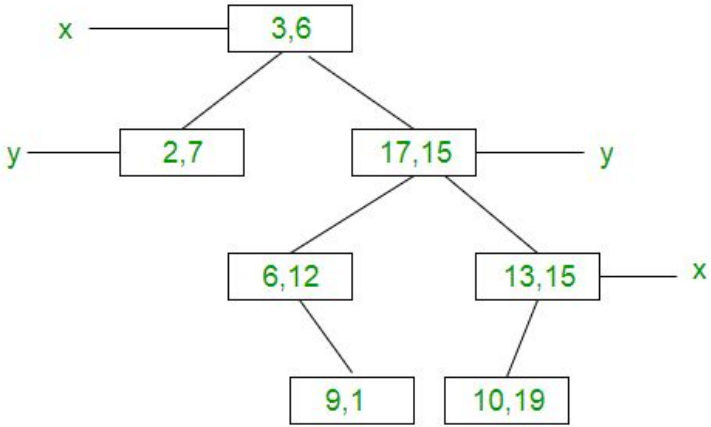
Insert (9, 1)

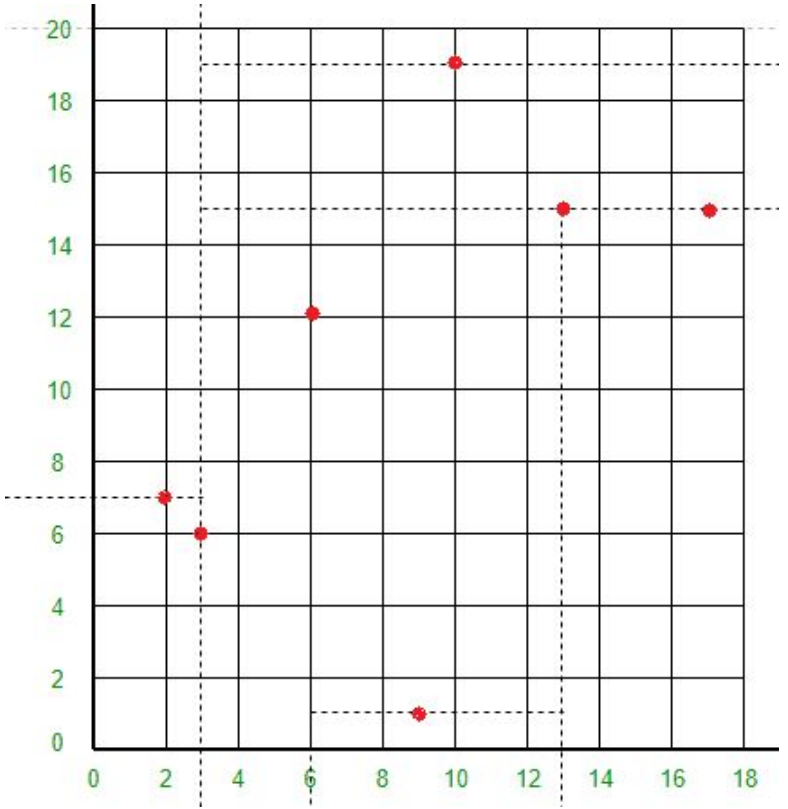




Creacion

Insert (10, 19)





Busqueda

```
bool searchRec(Node* root, int point[], unsigned depth)
{
    // Base cases
    if (root == NULL)
        return false;
    if (arePointsSame(root->point, point))
        return true;
    // Current dimension is computed using current depth and total
    // dimensions (k)
    unsigned cd = depth % k;
    // Compare point with root with respect to cd (Current dimension)
    if (point[cd] < root->point[cd])
        return searchRec(root->left, point, depth + 1);
    return searchRec(root->right, point, depth + 1);
}

bool search(Node* root, int point[])
{
    // Pass current depth as 0
    return searchRec(root, point, 0);
}
```

Encontrar el minimo

- Si la dimensión es la misma que la profundidad actual entonces el mínimo se encuentra en el lado izquierdo (si existe)
- Sino el mínimo puede ser en el lado izquierdo o derecho o el nodo actual asi que se escoge el mínimo de estos 3

Encontrar el minimo

```
int findMinRec(Node* root, int d, unsigned depth)
{
    // Base cases
    if (root == NULL)
        return INT_MAX;

    // Current dimension is computed using current depth and total
    // dimensions (k)
    unsigned cd = depth % k;

    // Compare point with root with respect to cd (Current dimension)
    if (cd == d)
        return min(root->point[d], findMinRec(root->left, d, depth + 1));


    // If current dimension is different then minimum can be anywhere
    // in this subtree
    return min(root->point[d],
               findMinRec(root->left, d, depth + 1),
               findMinRec(root->right, d, depth + 1));
}
```

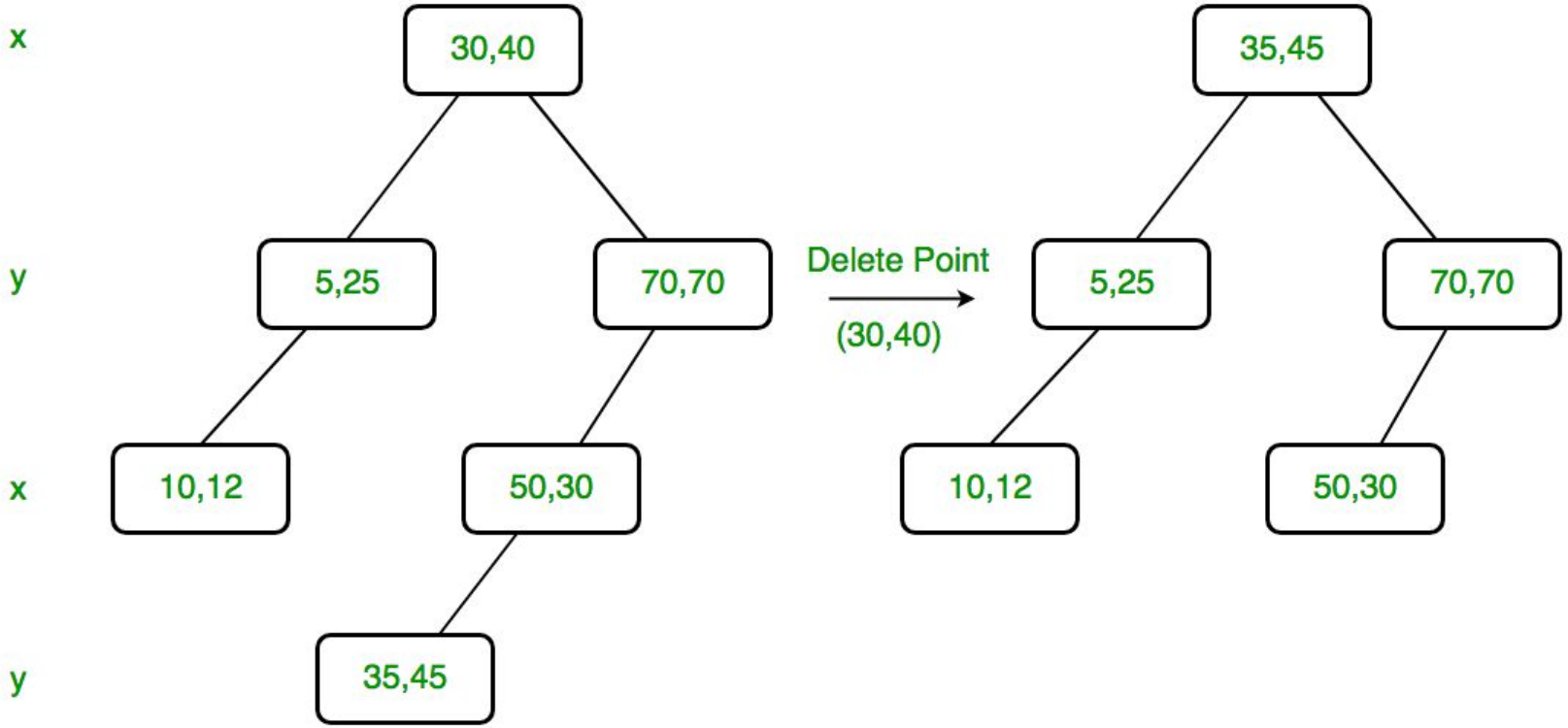
```
int findMin(Node* root, int d)
{

    return findMinRec(root, d, 0);
}
```

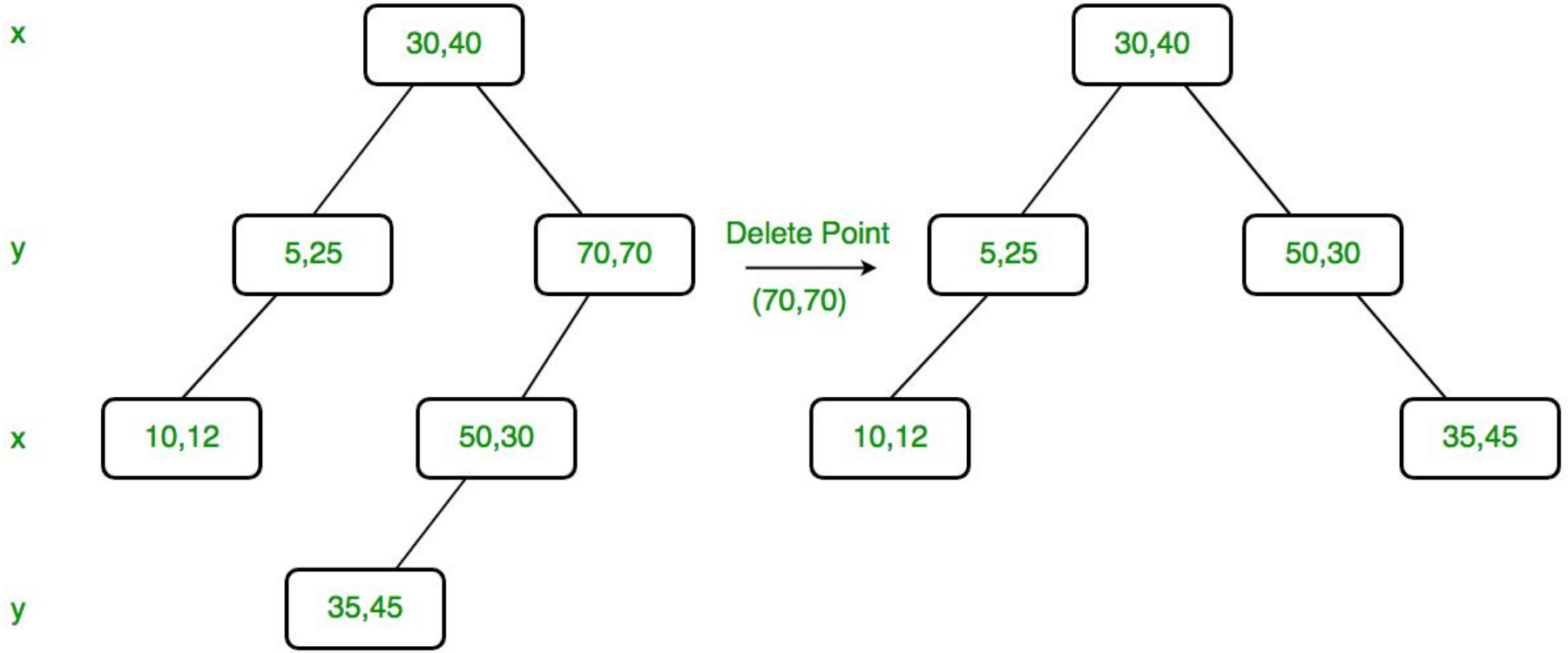
Suprimir

- Si el nodo actual es el nodo que queremos suprimir
  - Si no tiene hijos simplemente suprimir el nodo
  - Sino si el nodo tiene un subtree derecho
    - buscar el mínimo de su misma dimensión
    - reemplazar el nodo actual por el mínimo y suprimir el mínimo
  - sino si el nodo tiene un subtree izquierdo
    - buscar el mínimo de su misma dimensión
    - reemplazar el nodo actual por el mínimo y suprimir el mínimo
    - hacer del subtree izquierdo el subtree derecho
- Si el nodo actual es más grande que el elemento a suprimir buscar el nodo en el subtree izquierdo , derecho sino

Suprimir


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

Suprimir


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