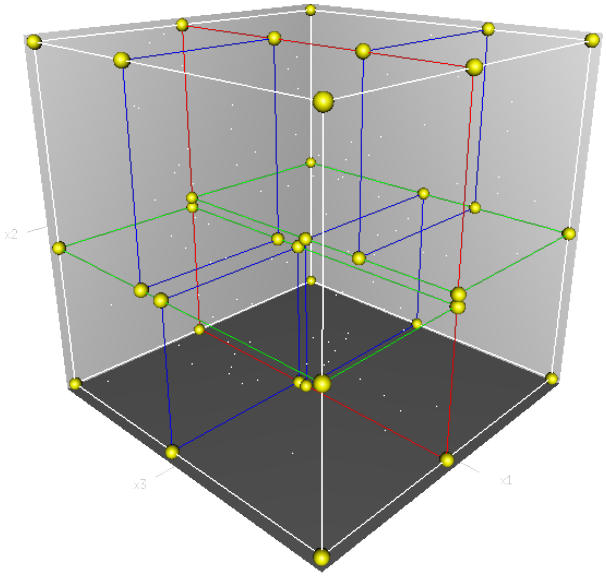
KD-Trees

Karim Pichara B.
Associate Professor
Computer Science Department
Pontificia Universidad Católica de Chile

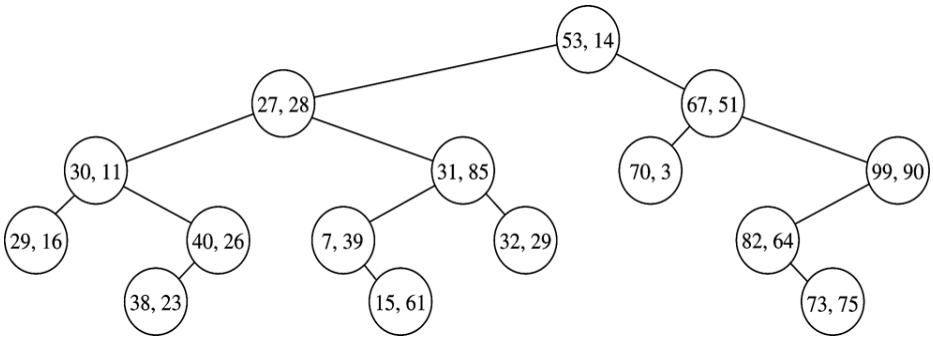
KD-Trees

- No es necesario recorrer todos los datos para encontrar los vecinos cercanos
- Usando una estructura para almacenar los datos podemos realizar una búsqueda más eficiente.
- Un KD-Tree es una estructura de datos que particiona el espacio de tal manera de organizar los puntos k-dimensionales

KD-Trees: 3D



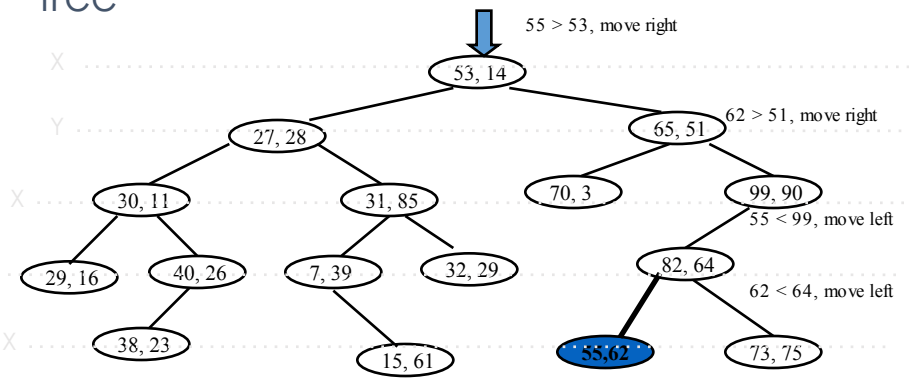
KD-Trees



2-D tree example

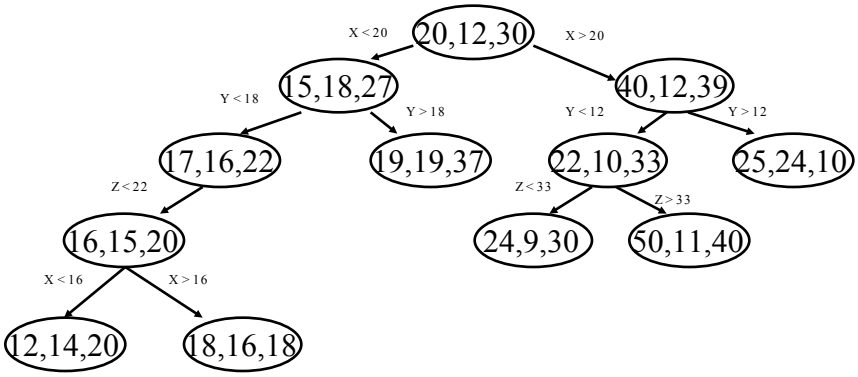
KD-Trees

Insert (55, 62) into the following 2-D tree



KD-Trees

3-D Example:

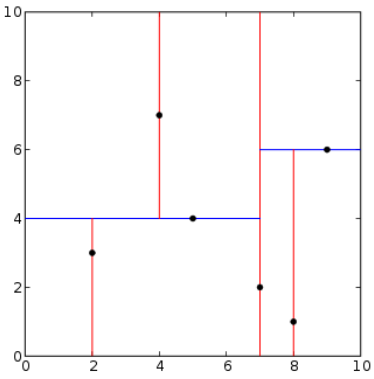


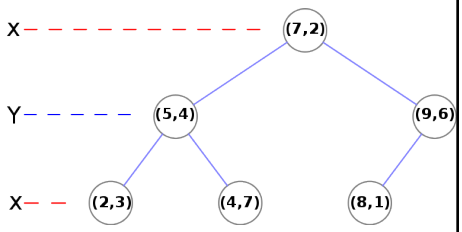
KD-Trees: Construction

```
Kd-Trees(object_list , depth = 0)
    if(object_list == empty)  return empty set

K = size(object_list ,2)      % Número de variables donde viven los datos
axis = mod(depth , k)    %Dimensión en cuestión según la profundidad

object_list.sort(axis)
largo = len(object_list)
median = object_list(largo// 2)

Node = []
Node.location = median
Node.left_child = Kd-Trees(object_list(0:median),  depth+1)
Node.right_child = Kd-Trees(object_list(median+1:largo),  depth+1)

return Node
```

KD-Trees: Construction Example

Object List = (2,3) (5,4) (9,6) (4,7) (8,1) (7,2)





Karim Pichara B.

1- NN search Algorithm

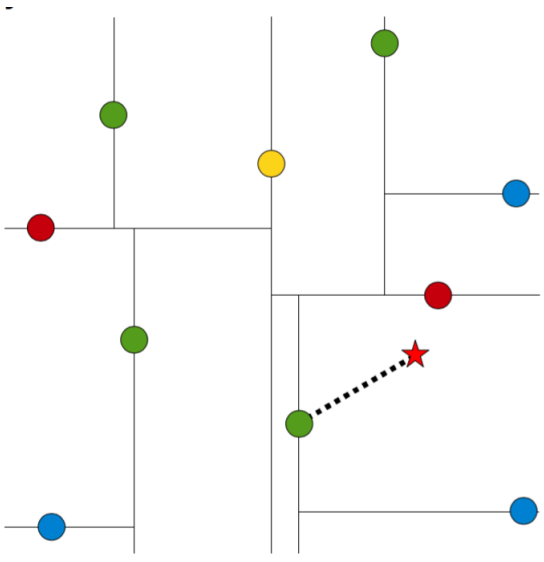
1. Starting from the root move down the three recursively (like in the construction process)

2. Once a leaf node is reached:
   current best = leaf
   current distance = dist(leaf,x)

3. Unwind the recursion, move one level up:
   - if the current node is closer than the current best
     current best = current node
     current distance = dist(current node, x)

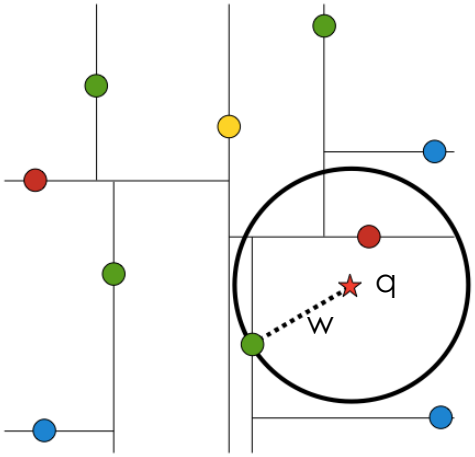
   - if the other child's size could be a closer point to x than the current best
     KNN-KD-tree(KD_tree(root = current node), x)
   else
     Go to step 3

KD-Trees: NN search



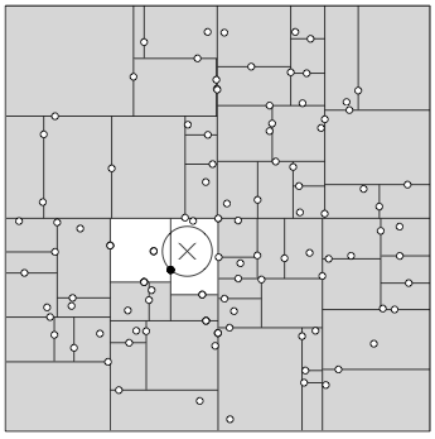
KD-Trees: NN search

if q[axis] – best_distance <= node
if q[axis] + best_distance > node



W= best distance

KD-Trees: NN search



good case

bad case