PRACTICA 6 DE LABORATORIO

DOCENTE	CARRERA	CURSO
Vicente Machaca Arceda	Escuela Profesional de Ciencias de la Computación	Estructuras de Datos Avanzadas

PRÁCTICA N°	TEMA	DURACIÓN (HORAS)
06	K-d Tree	10

1. OBJETIVOS

- Implementar la estructura K-d Tree

2. TEMAS A TRATAR

- K-d Tree

3. MARCO TEÓRICO

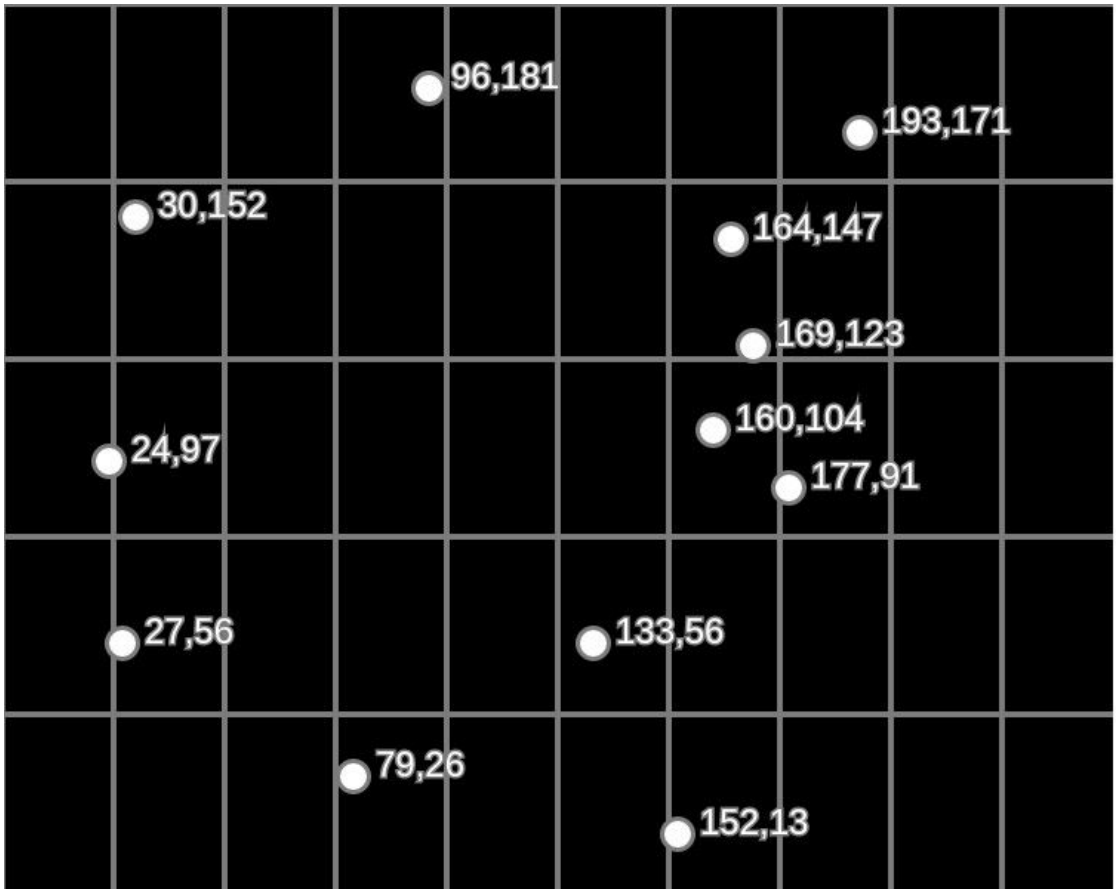
K-D TREE
(Extraido de: https://www.geeksforgeeks.org/k-dimensional-tree/)

A K-D Tree(also called as K-Dimensional Tree) is a binary search tree where data in each node is a K-Dimensional point in space. In short, it is a space partitioning(details below) data structure for organizing points in a K-Dimensional space.

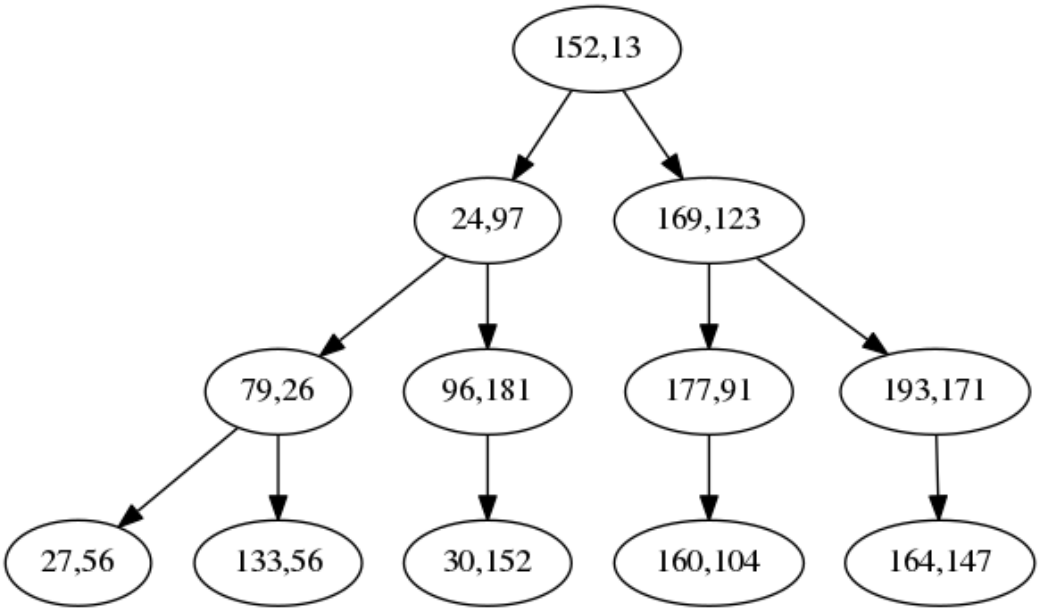
A non-leaf node in K-D tree divides the space into two parts, called as half-spaces.

Points to the left of this space are represented by the left subtree of that node and points to the right of the space are represented by the right subtree. We will soon be explaining the concept on how the space is divided and tree is formed.

For example for the next points:



We build the next tree:



4. EJERCICIOS

We will develop the search method in K-d Tree:

1. **With the last k-d tree implemented in javascript, complete the *closest_point_brute_force* and *naive_closest_point* functions.**

```
k = 2;

class Node{
    constructor(point, axis){
        this.point = point;
        this.left = null;
        this.right = null;
        this.axis = axis;
    }
}

function distanceSquared(point1, point2){
    var distance = 0;
    for (var i = 0; i < k; i++)
        distance += Math.pow((point1[i] - point2[i]), 2);
    return Math.sqrt(distance);
}

function closest_point_brute_force(points, point){}
function naive_closest_point(node, point, depth = 0, best = null){
//algorithm
    //1. best = min(distance(point, node.point), best)
    //2. chose the branch according to axis per level
    //3. recursevely call by branch chosed
}
```

2. **Evaluate your results with this data and point (compare the results of the two functions developed before):**

```
var data = [
        [40,70],
        [70,130],
        [90,40],
```

```
        [110, 100],
        [140,110],
        [160, 100]
    ];
var point = [140,90];
```

3. **Now, evaluate your results with this data and point (compare the results of the two functions developed before):**

```
var data = [
        [40,70],
        [70,130],
        [90,40],
        [110, 100],
        [140,110],
        [160, 100],
        [150, 30]
    ];
var point = [140,90];
```

4. **Complete the function *closest_point* in order to get the closest point:**

```
function closest_point(node, point, depth = 0){
    //algorithm
    //1. set "next_branch" and "opposite_branch" to look for according to axis
and level
    //2. chose "best" distance between (node.point, next_branch, point)
    //3. If (distance(point, "best") > abs(point[axis] - node.point[axis])
    //4.      chose "best" distance between (node.point, opposite_branch,
point)
    //5. return best


    if (node === null)
        return null;

    var axis = depth % k;
    var next_branch = null; //next node brach to look for
    var opposite_branch = null; //opposite
 node brach to look for
```

```
    if (point[axis] < node.point[axis]){
        next_branch = node.left;
        opposite_branch = node.right;
    }else{
        next_branch = node.right;
        opposite_branch = node.left;
    }

    //YOUR CODE HERE

    return best;
}
```

5. **Develop a search function in order to get the *n* nearest points**

REFERENCIAS

Maneewongvatana, S., & Mount, D. M. (1999, December). It's okay to be skinny, if your friends are fat. In *Center for Geometric Computing 4th Annual Workshop on Computational Geometry* (Vol. 2, pp. 1-8).