

Práctica 04

DOCENTE	CARRERA	CURSO
Vicente Machaca Arceda	Maestría en Ciencia de la Computación	Algoritmos y Estructura de Datos

PRÁCTICA	TEMA	DURACIÓN
04	Algoritmos de ordenamiento	3 horas

1. Datos de los estudiantes

Grupo: N° 8

■ Integrantes:

- Esai Josue Huaman Meza
- Alan Jerry Reyes Robles
- Jorge Luis Zegarra Guardamino
- Nestor Giraldo Calcinas Huaranga

2. Introducción

Para esta Práctica, se implementará la estructura multidimensional KD-Tree.

Para esto se usará los códigos dados por el docente en la práctica y así mismo se terminarán de colocar algunos algoritmos faltantes. Luego de ello, se modificará para dos determinadas coordenadas y se mostrarán en index.html y main.html los resultados.

Para todo ello, el lenguaje de programación a usar será Java Script.

El repositorio Github se encuentra en el siguiente enlace Estructura KD-Tree. El video explicativo se encuentra en el siguiente enlace Algoritmo Multidimensional KD-Tree-Grupo8.

3. Estructuras de Datos Multidimensional

1. Estructura de datos KD-Tree

La estructura KD-Tree es una estructura de datos de particionado del espacio que organiza los puntos en un Espacio euclídeo de k dimensiones.

La Estructura KD-Tree se puede construir de la siguiente manera:

- Conforme se desciende en el árbol, se emplean ciclos a través de los ejes para seleccionar los planos.
- En cada paso, el punto seleccionado para crear el plano de corte será la mediana de los puntos puestos en el árbol kd, lo que respeta sus coordenadas en el eje que está siendo usado.

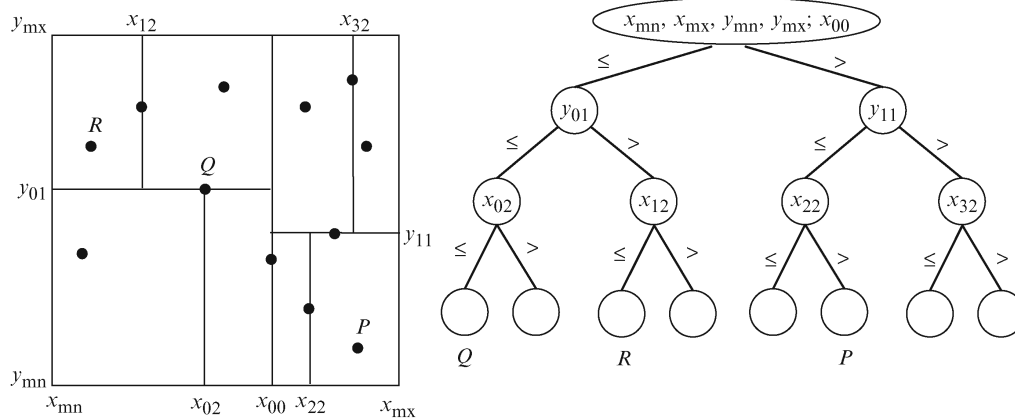


Figura 1: Estructura KD-Tree

4. Implementación

Se desarrolló la estructura KDTree implementando los cambios solicitados en la práctica, los cuales se pueden encontrar en el siguiente repositorio Github Estructura KD-Tree, y se obtienen las siguientes imágenes.

5. Resultados

1. Estructura KD-Tree

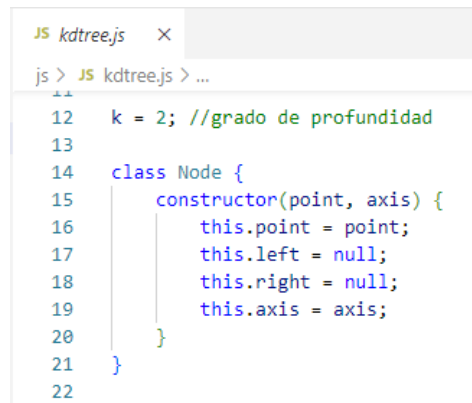
- Crear un archivo main.html

```

<> main.html x
<> main.html > html
1  <html>
2
3  <head>
4
5      <title>Kd tree</title>
6      <script src= "js/p5.min.js"> </script>
7      <script src= "js/kdtree.js"> </script>
8      <script src= "js/sketch.js"> </script>
9
  
```

Figura 2: Archivo main.html

- Crear un archivo kdtree.js



```
JS kdtree.js x
js > JS kdtree.js > ...
11
12 k = 2; //grado de profundidad
13
14 class Node {
15     constructor(point, axis) {
16         this.point = point;
17         this.left = null;
18         this.right = null;
19         this.axis = axis;
20     }
21 }
22
```

Figura 3: Archivo kdtree.js

- Construir function getHeight(node) en kdtree.js

```
23 //Retorna la altura del arbol.
24 function getHeight(node) {
25     if (node === null) {
26         return 0;
27     } // Encuentra la altura de cada rama: izq y der
28     var lh = getHeight(node.left);
29     var rh = getHeight(node.right);
30     return 1 + Math.max(lh, rh);
31 }
32
```

Figura 4: function getHeight(node)

- Construir function generate dot(node) en kdtree.js

```
33 //Genera al arbol en formato dot, por ejemplo:
34 function generate_dot(node) {
35     // alert("prueba");
36     if (node === null) {
37         return "";
38     }
39     var tmp = '';
40     if (node.left != null) {
41         tmp += ' ' + node.point.toString() + ' -> ' + ' ' + node.left.point.toString() + ' ' + '\n';
42         tmp += generate_dot(node.left);
43     }
44     if (node.right != null) {
45         tmp += ' ' + node.point.toString() + ' -> ' + ' ' + node.right.point.toString() + ' ' + '\n';
46         tmp += generate_dot(node.right);
47     }
48
49     return tmp;
50 }
```

Figura 5: function generate dot(node)

- Construir function build kdtree en kdtree.js

```
52 //Construye el KD-Tree y retorna el nodo raiz.
53 function build_kdtree(points, depth = 0) {
54     var n = points.length;
55     var axis = depth % k;
56
57     if (n <= 0) {
58         return null;
59     }
60     if (n == 1) {
61         return new Node(points[0], axis)
62     }
63
64     var median = Math.floor(points.length / 2);
65
66     // sort by the axis
67     points.sort(function (a, b) {
68         return a[axis] - b[axis];
69     });
70
71     var left = points.slice(0, median);
72     var right = points.slice(median + 1);
73
74
75     var node = new Node(points[median].slice(0, k), axis);
76     node.left = build_kdtree(left, depth + 1);
77     node.right = build_kdtree(right, depth + 1);
78
79     return node;
80 }
```

Figura 6: function build kdtree

- Crear un archivo sketch.js

```
JS sketch.js x
js > JS sketch.js > ...
1  var root
2  var width
3  var height
4  var octx
5  var ocanvas
6
7  points = [
8      [40, 70],           // var data = [
9      [70, 130],          // [40 ,70] ,
10     [90, 40],            // [70 ,130] ,
11     [110, 100],          // [90 ,40] ,
12     [140, 110],          // [110 , 100] ,
13     [160, 150]           // [140 ,110] ,
14 ];                      // [160 , 100]
15                         // ];
16 function setup() {
17     width = 500;
18     height = 400;
19     let kdtreeCanvas = createCanvas(width, height);
20     kdtreeCanvas.parent("kdtreeCanvas");
21 }
```

Figura 7: Archivo sketch.js

■ Digraph G

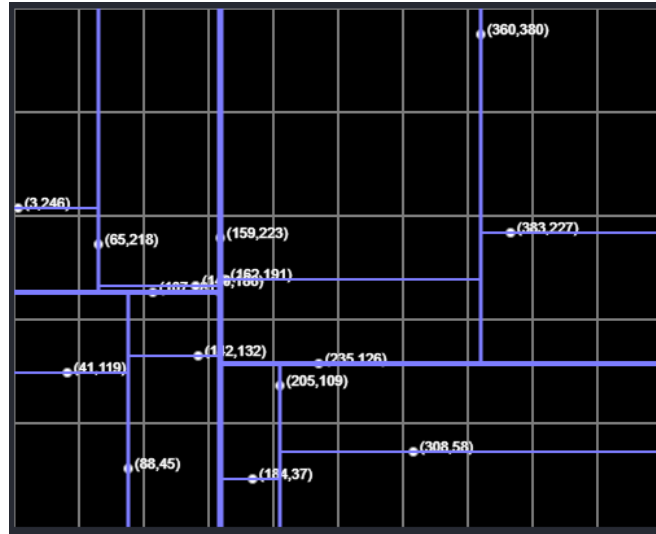


Figura 8: KD-Tree ejemplo

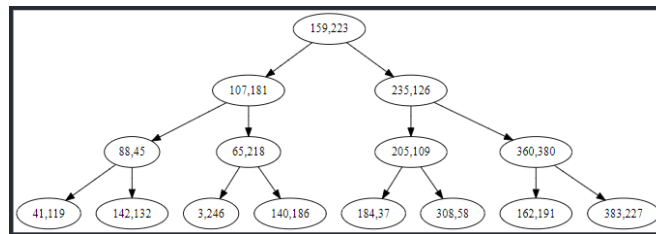


Figura 9: KD-Tree ejemplo

Obtener Altura del KDTree: 4	sketch.js:58
Punto mas cercano por Naive Closest Point:	sketch.js:61
Generacion de Dot:	sketch.js:62

```

digraph G {
  "159,223" -> "107,181";
  "107,181" -> "88,45";
  "88,45" -> "41,119";
  "88,45" -> "142,132";
  "107,181" -> "65,218";
  "65,218" -> "3,246";
  "65,218" -> "140,186";
  "159,223" -> "235,126";
  "235,126" -> "205,109";
  "205,109" -> "184,37";
  "205,109" -> "308,58";
  "235,126" -> "360,380";
  "360,380" -> "162,191";
  "360,380" -> "383,227";
}

```

Figura 10: KD-Tree ejemplo

- Implemente la función closest point brute force

```
90 function closest_point_brute_force(points, point) {
91     var distance = null;
92     var best_distance = null;
93     var best_point = null;
94     for (let i = 0; i < points.length; i++) {
95         distance = distanceSquared(points[i], point);
96         // console.log(distance);
97         if (best_distance === null || distance < best_distance) {
98             best_distance = distance;
99             //best_point = { 'point': points[i], 'distance': distance }
100             best_point = points[i];
101         }
102     }
103     return best_point;
104 }
```

Figura 11: función closest point brute force

- Implemente la función naive closest point

```
107 function naive_closest_point(node, point, depth = 0, best = null) {
108     //algorithm
109     //1. best = min(distance(point, node.point), best)
110     //2. chose the branch according to axis per level
111     //3. recursevely call by branch chosed
112     if (node === null)
113         return best;
114     var axis = depth % k;
115
116     // if (point[axis] < node.point[axis])
117     // console.log("axis",point[axis])
118     // console.log("node axis",node.point[axis][1])
119
120
121     var next_best = null; //next best point
122     var next_branch = null; //next node brach to look for
123     if (best === null || (distanceSquared(best, point) > distanceSquared(node.point, point)))
124         next_best = node.point;
125     else
126         next_best = best;
127     // if (point[axis] < node.point[axis])
128     if (point[axis] < node.point[axis])
129         next_branch = node.left
130     else
131         next_branch = node.right
132     return naive_closest_point(next_branch, point, depth + 1, next_best);
133 }
```

Figura 12: función naive closest point

- value el resultado de las dos funciones implementadas anteriormente con este conjunto de datos: [40,70], [70,130], [90,40], [110, 100], [140,110], [160, 100]

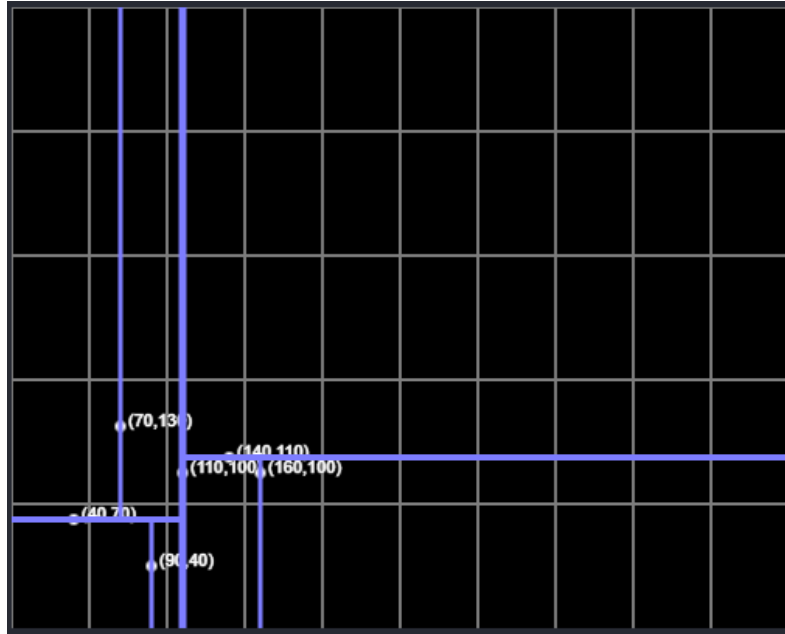


Figura 13: Visualización KD-Tree

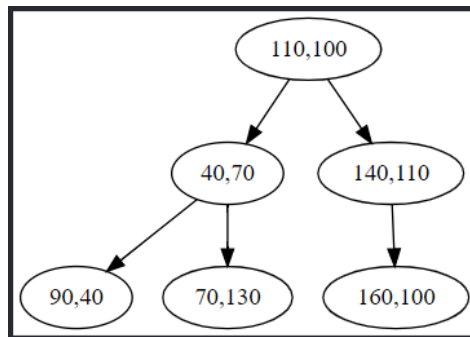


Figura 14: Visualización KD-Tree

```
Obtener Altura del KDTree: 3
Punto mas cercano por Naive Closest Point:
Generacion de Dot:
digraph G {
  "110,100" -> "40,70";
  "40,70" -> "90,40";
  "40,70" -> "70,130";
  "110,100" -> "140,110";
  "140,110" -> "160,100";
}
```

Figura 15: Visualización de la Consola del Navegador

- value el resultado de las dos funciones implementadas anteriormente con este conjunto de datos: [40,70], [70,130], [90,40], [110,100], [140,110], [160,100], [150,30]

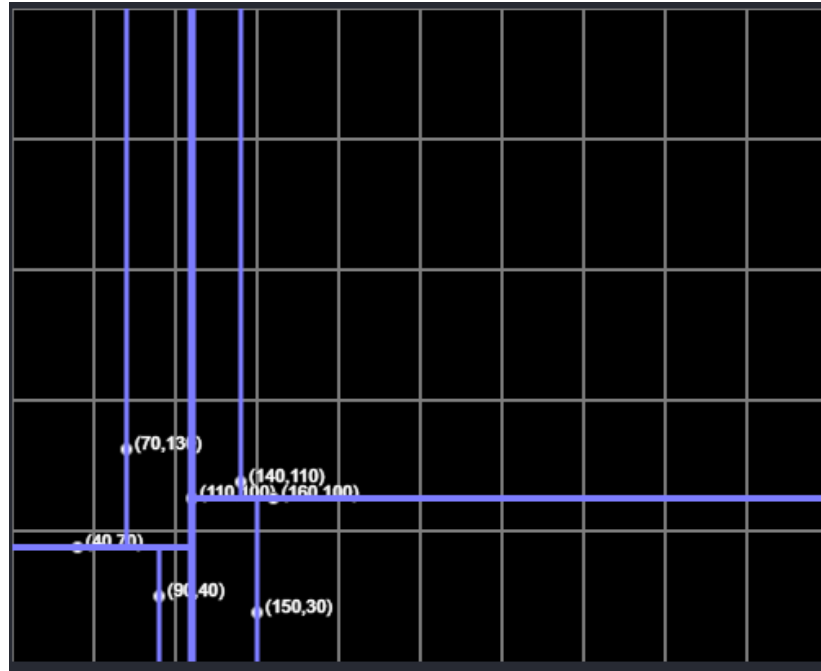


Figura 16: Visualización KD-Tree

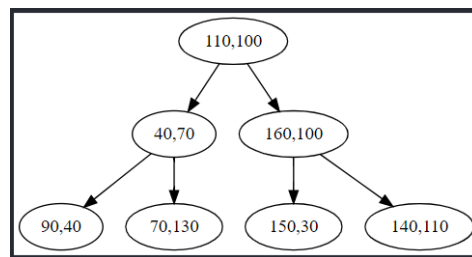


Figura 17: Visualización KD-Tree

```
Obtener Altura del KDTree: 3
Punto mas cercano por Naive Closest Point:
Generacion de Dot:
digraph G {
  "110,100" -> "40,70";
  "40,70" -> "90,40";
  "40,70" -> "70,130";
  "110,100" -> "160,100";
  "160,100" -> "150,30";
  "160,100" -> "140,110";
}
```

Figura 18: Visualización de la Consola del Navegador

- Implemente la función closest point

```
148 function closest_point(node, point, depth = 0) {
149     // 1. Set next_branch and opposite_branch to look for according to axis and level
150     // 2. Chose best distance between (node.point, next_branch, point)
151     // 3. if (distance(point, best) > abs(point[axis] - node.point[axis]))
152     // 4. chose best distance between (node.point, opposite_branch, point)
153
154     if (node == null)
155         return null;
156     // best = min(distanceSquared(node.point, point));
157     var axis = depth % k;
158     var next_branch = null; // next node branch to look for
159     var opposite_branch = null; // opposite node branch to look for
160
161     if (point[axis] < node.point[axis]) {
162         next_branch = node.left;
163         opposite_branch = node.right;
164     } else {
165         next_branch = node.right;
166         opposite_branch = node.left;
167     }
168     var best = closer_point(point, closer_point(point, closest_point(next_branch, point, depth + 1), node), best);
169
170     if (distanceSquared(best.point, point) > Math.abs(point[node.axis] - node.point[axis])) {
171         best2 = closer_point(point, closest_point(opposite_branch, point, depth + 1), node);
172     }
173     best = closer_point(point, best2, best);
174
175     return best;
176 }
177
```

Figura 19: función closest point

- Implemente la función KNN

```
159 function KNN(node, point_2, cant_puntos) {
160     var closest_points = [];
161     for (var i = cant_puntos - 1; i >= 0; i--) {
162         punto = closest_point(node, point_2);
163         if (punto == null) continue;
164         punto.estado = false;
165         closest_points.push(punto);
166     }
167     var ans = []
168     for (var i = closest_points.length - 1; i >= 0; i--) {
169         punto = closest_points[i];
170         punto.estado = true;
171         ans.push(punto.point);
172     }
173     return ans;
174 }
175
```

Figura 20: función KNN

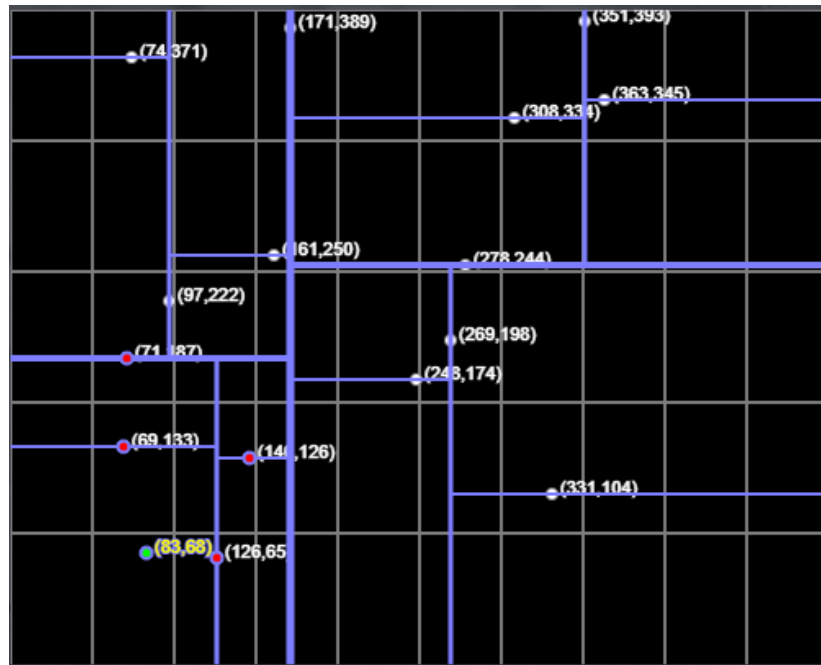


Figura 21: función KNN ejemplo

- Implemente la función range query circle

```

177 function range_query_circle(node, center, radio, queue, depth = 0) {
178   if (node == null) return null;
179
180   var axis = node.axis;
181   var nb = null;
182   var ob = null;
183
184   if (center[axis] < node.point[axis]) {
185     nb = node.left;
186     ob = node.right;
187   } else {
188     nb = node.right;
189     ob = node.left;
190   }
191
192   var best = closer_point(center, node, range_query_circle(nb, center, radio, queue, depth + 1));
193
194   if (Math.abs(center[axis] - node.point[axis]) <= radio || distanceSquared(center, best.point) > Math.abs(ce
195     if (distanceSquared(center, node.point) <= radio) {
196       queue.push(node.point);
197     }
198     best = closer_point(center, best, range_query_circle(ob, center, radio, queue, depth + 1));
199   }
200   return best;
201 }

```

Figura 22: función range query circle

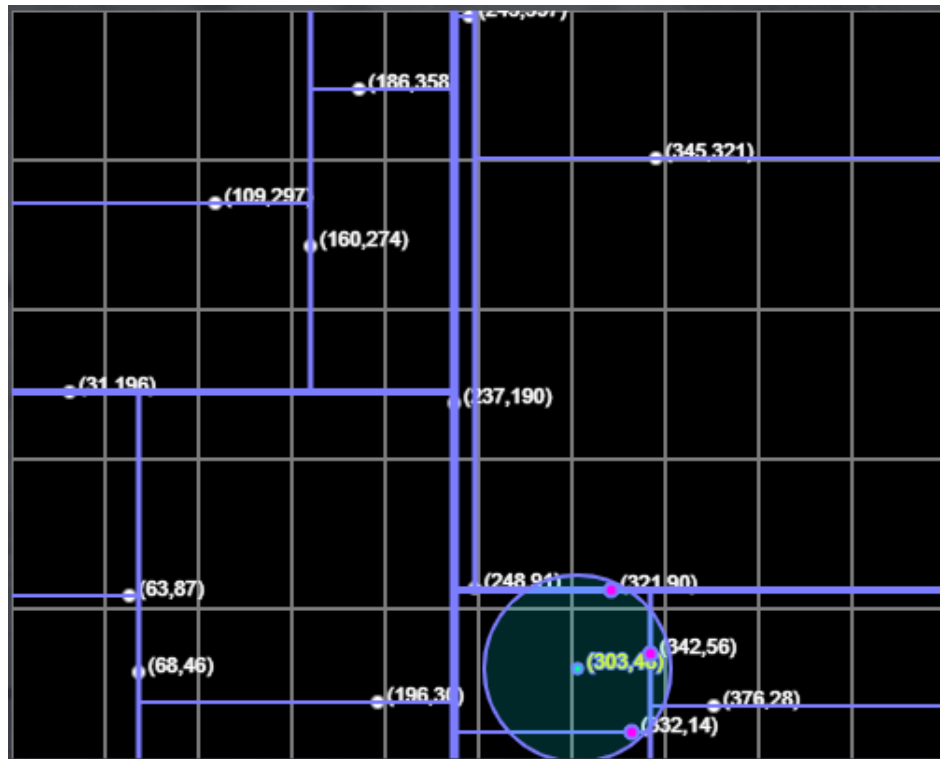


Figura 23: función range query circle ejemplo

- Implemente la función range query rec

```

203 function range_query_rect(node, center, hug, queue, depth = 0) {
204     if (node == null) return null;
205
206     var axis = node.axis;
207     var nb = null;
208     var ob = null;
209
210     if (center[axis] < node.point[axis]) {
211         nb = node.left;
212         ob = node.right;
213     } else {
214         nb = node.right;
215         ob = node.left;
216     }
217     var best = closer_point(center, node, range_query_rect(nb, center, hug, queue, depth + 1));
218
219     if (Math.abs(center[axis] - node.point[axis]) <= hug[axis] * 2 || distanceSquared(center, best.point) > Math
220
221         if (Math.abs(center[0] - node.point[0]) <= hug[0] && Math.abs(center[1] - node.point[1]) <= hug[1]) {
222             queue.push(node.point);
223         }
224         best = closer_point(center, best, range_query_rect(ob, center, hug, queue, depth + 1));
225     }
226
227     return best;
228 }
229

```

Figura 24: función range query rec

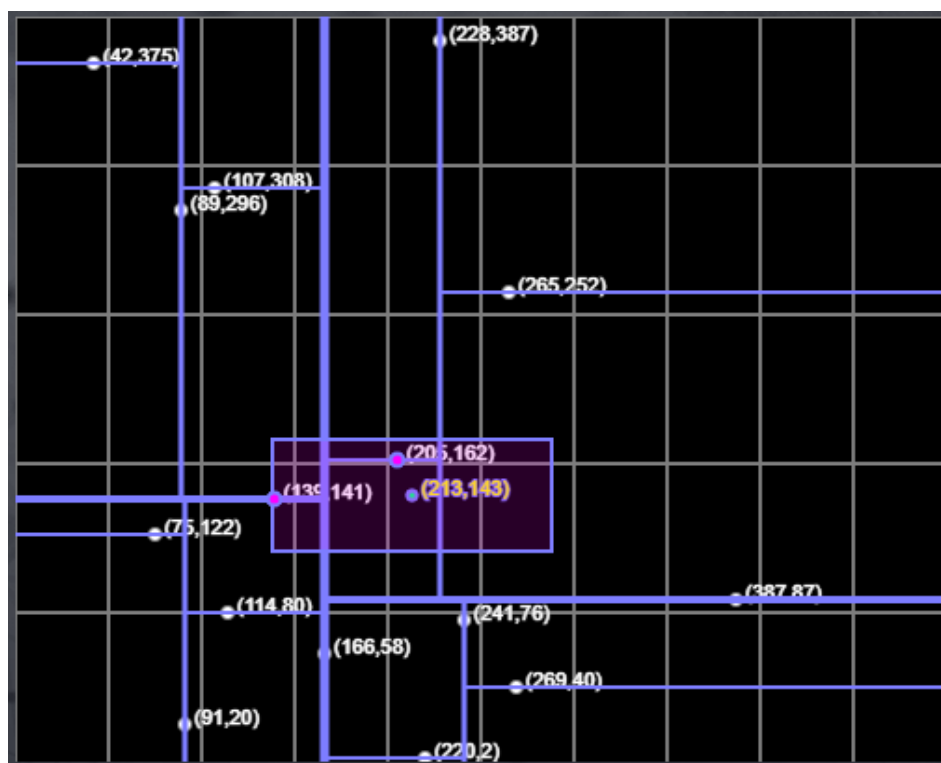


Figura 25: función range query rec ejemplo

6. Conclusiones

- Se comprueba que mediante la implementación del Kdtree como estructura multidimensional, reduce el número de vecinos a buscar, al calcular la distancia del punto objetivo, técnica utilizada en el algoritmo KNN para clasificación en Machine learning.
- El rendimiento de la estructura de datos es inversamente proporcional a la cantidad de los datos, y a su vez, la dimensionalidad del árbol, ya que al incrementar los datos y la dimensionalidad, se irá reduciendo el rendimiento, ya que cuando se vuelve muy denso, puede haber intersecciones indeseadas con vecinos muy cercanos.
- La implementación de algoritmo KD-Tree con tecnologías Html5, JavaScript. P5, ha sido muy intuitivo para comprender el comportamiento de datos multidimensional.