Universidad de Nacional de San Agustín
Facultad de Producción y Servicios
Escuela de Ciencia de la Computación

Practica 7

KD - Tree

Integrantes: Mita Yagua Lesly Yaneth
Pinto Medina Brian
Ccari Quispe José
Gonza Condori, Gabriel
Flores Herrera, Jefferson
Profesor: Msc. Vicente Machaca Arceda

Fecha de entrega: 11 de noviembre de 2020
Arequipa, Perú

Índice de Contenidos

Índice de Figuras

Índice de Códigos

1. Range Query Circle

Se incluyo en el archivo **kdtree.js** la función *range_query_circle*. La captura de pantalla del **Código 4** se encuentra en la sección Capturas de pantalla como **Figura 4**.
A continuación se muestra la función implementada en el siguiente código y los resultados de este como visualización web.

Código 1: range_query_circle

```
function range_query_circle(data , center , radio , queue , depth = 0) {
    let neight = [];
    let root = buildKDTree(data);

    for(let i = 0; i < data.length; ++i) {
        let arr = [];
        convertKDTreeToArray(root, arr);
        let closePoint = closest_point(root, center,depth);
        if(distanceSquared( center,closePoint.point.vectorialSpace) < radio){
            neight.push(closePoint.point.vectorialSpace);
        }
        deleteNode(arr, closePoint);
        root = buildKDTree(arr);
    }
    return neight;
}
```

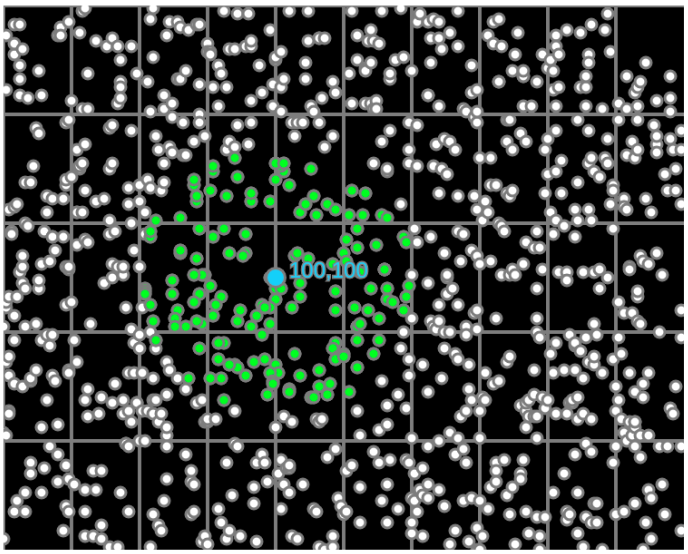

Figura 1: Visualización del navegador Web Circulo

2. Range Query Rec

Se incluyo en el archivo **kdtree.js** la función *range_query_rec*. La captura de pantalla del **Código 4** se encuentra en la sección Capturas de pantalla como **Figura 4**.
A continuación se muestra la función implementada en el siguiente código y los resultados de este como visualización web.

Código 2: range_query_rec

```
function range_query_circle(data , center , radio , queue , depth = 0) {
    let neight = [];
    let root = buildKDTree(data);

    for(let i = 0; i < data.length; ++i) {
        let arr = [];
        convertKDTreeToArray(root, arr);
        let closePoint = closest_point(root, center,depth);
        if(distanceSquared( center,closePoint.point.vectorialSpace) < radio){
            neight.push(closePoint.point.vectorialSpace);
        }
        deleteNode(arr, closePoint);
        root = buildKDTree(arr);
    }
    return neight;
}
```

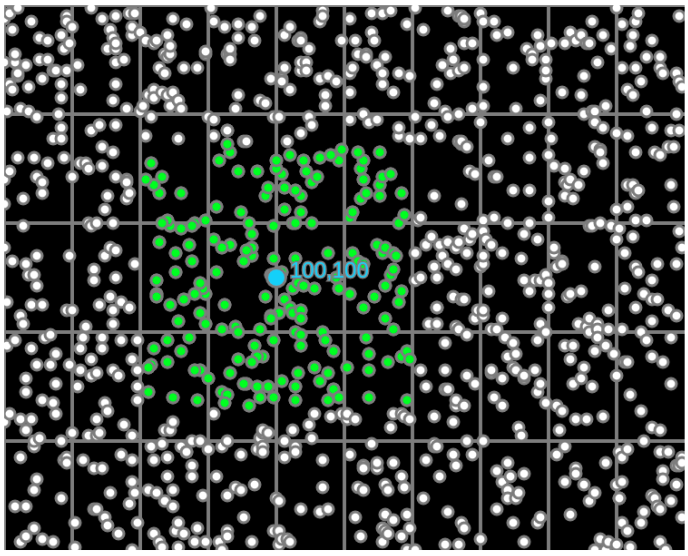

Figura 2: Visualización del navegador Web Rectangulo

3. Código fuente

Código 3: main.html

```
<html>
    <head>
        <title>KD Tree</title>
        <script src = "p5.min.js"></script>
        <script src = "kdtree.js"></script>
        <script src = "sketch.js"></script>
    </head>
    <body>

    </body>
</html>
```

Código 4: kdtree.js

```
k = 2;

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

class N_Point {
    constructor(points) {
        this.vectorialSpace = points;
    }
}

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 ) {

    var closestPoint ;
    var minDistance;
    var distance;
    for(var i= 0; i< points.length ;i++){
        if(i==0){
            minDistance = distanceSquared(points[0].vectorialSpace,point);
            closestPoint = points[0]. vectorialSpace;
```

```
34          }
35          distance = distanceSquared(points[i].vectorialSpace,point);
36
37          if(minDistance >= distance ){
38              minDistance = distance;
39              closestPoint = points[i].vectorialSpace;
40          }
41      }
42      return  closestPoint;
43  }
44
45  function naive_closest_point(node , point , depth = 0, best = null ) {
46      if(node != null){
47          if(depth == 0){
48              best = node.point.vectorialSpace;
49          }
50
51          var axisDistance = point[node.axis] - node.point.vectorialSpace[node.axis];
52          let distanceBest = distanceSquared(best, point);
53          let distanceNode = distanceSquared(node.point.vectorialSpace,point);
54
55
56          if ( Math.abs(axisDistance) <= distanceBest ){
57              if(distanceBest > distanceNode){
58                  best = node.point.vectorialSpace;
59              }
60              if(axisDistance > 0){
61                  return naive_closest_point (node.right , point , depth+1, best );
62              }
63              else{
64                  return naive_closest_point (node.left , point , depth+1, best);
65              }
66          }else{
67              return best;
68          }
69      }else{
70          return best;
71      }
72  }
73
74  ////////////////////Practica-07  ////////////////////////////////////////
75
76
77  function range_query_circle(data , center , radio , queue , depth = 0) {
78      let neight = [];
79      let root = buildKDTree(data);
80
81      for(let i = 0; i < data.length; ++i) {
82          let arr = [];
83          convertKDTreeToArray(root, arr);
84          let closePoint = closest_point(root, center,depth);
85          if(distanceSquared( center,closePoint.point.vectorialSpace) < radio){
```

```
86              neight.push(closePoint.point.vectorialSpace);
87          }
88          deleteNode(arr, closePoint);
89          root = buildKDTree(arr);
90      }
91      return neight;
92  }
93
94  function range_query_rec(data , center , diameter , queue , depth = 0) {
95      let neight = [];
96      let root = buildKDTree(data);
97
98      for(let i = 0; i < data.length; ++i) {
99          let arr = [];
100         convertKDTreeToArray(root, arr);
101         let closePoint = closest_point(root, center,depth);
102
103         let dimX = [closePoint.point.vectorialSpace [0],0];
104         let centerX =[center [0],0];
105
106         let dimY = [0,closePoint.point.vectorialSpace [1]];
107         let centerY =[0,center [1]];
108
109         if(distanceSquared(centerX,dimX) < diameter && distanceSquared(centerY,dimY) <
    ↪ diameter){
110             neight.push(closePoint.point.vectorialSpace);
111         }
112         deleteNode(arr, closePoint);
113         root = buildKDTree(arr);
114     }
115     return neight;
116 }
117 ////////////////////////////////////////////////////////
118
119
120 function closest_point(node, point, depth = 0) {
121     if(node === null) {
122         return null;
123     }
124
125     if(point[node.axis] < node.point.vectorialSpace[node.axis]) {
126         var nextBranch = node.left;
127         var otherBranch = node.right;
128     } else {
129         var nextBranch = node.right;
130         var otherBranch = node.left;
131     }
132
133     var temp = closest_point(nextBranch, point, depth + 1);
134     var best = closest(temp, node, point);
135
136     var distanceBest = distanceSquared(point, best.point.vectorialSpace);
```

```
137     var distanceAxis = Math.abs(point[node.axis] - node.point.vectorialSpace[node.axis]);
138 
139     if(distanceAxis <= distanceBest) {
140         temp = closest_point(otherBranch, point, depth + 1);
141         best = closest(temp, best, point);
142     }
143 
144     return best;
145 }
146 
147 function closest(node, root, point) {
148     if(node == null)
149         return root;
150     if(root == null)
151         return node;
152 
153     let distanceNode = distanceSquared(node.point.vectorialSpace, point);
154     let distanceRoot = distanceSquared(root.point.vectorialSpace, point);
155 
156     if(distanceNode < distanceRoot)
157         return node;
158     else
159         return root;
160 }
161 
162 function convertKDTreeToArray(node,array){
163     array.push(node.point);
164     if(node.left != null)
165         convertKDTreeToArray(node.left,array);
166     if(node.right != null)
167         convertKDTreeToArray(node.right,array);
168 }
169 
170 function KNN(data, n, point) {
171     let neight = [];
172     let root = buildKDTree(data);
173 
174     for(let i = 0; i < n; ++i) {
175         let arr = [];
176         convertKDTreeToArray(root, arr);
177         let closePoint = closest_point(root, point);
178         neight.push(closePoint.point.vectorialSpace);
179         deleteNode(arr, closePoint);
180         root = buildKDTree(arr);
181     }
182     return neight;
183 }
184 
185 function deleteNode(arr, node) {
186     for(let i = 0; i < arr.length; ++i) {
187         if(arr[i].vectorialSpace == node.point.vectorialSpace)
188             arr.splice(i, 1);
```

```
189     }
190 }
191
192 function getHeight(node) {
193     if(node === null)
194         return 0;
195
196     return Math.max(getHeight(node.left), getHeight(node.right)) + 1;
197 }
198
199 function generateDot(node) {
200     var s = "digraph G{\n";
201     var cola = [];
202     cola.push(node);
203     while(cola.length > 0){
204         let nodo = (cola.splice(0, 1))[0];
205         if(nodo.left == null){
206             continue;
207         }
208         let space = nodo.point.vectorialSpace.length;
209         for(let i = 0; i < space - 1; ++i){
210             s += "\"";
211             s += nodo.point.vectorialSpace[i];
212             s += ",";
213         }
214         s += nodo.point.vectorialSpace[space - 1];
215         s += "\" -> ";
216         for(let i = 0; i < space - 1; ++i){
217             s += "\"";
218             s += nodo.left.point.vectorialSpace[i];
219             s += ",";
220         }
221         s += nodo.left.point.vectorialSpace[space -1];
222         s += "\";\n";
223         cola.push(nodo.left);
224
225         //RIGHT NODE
226         if(nodo.right == null){
227             continue;
228         }
229         for(let i = 0; i < space - 1; ++i){
230             s += "\"";
231             s += nodo.point.vectorialSpace[i];
232             s += ",";
233         }
234         s += nodo.point.vectorialSpace[space - 1];
235         s += "\" -> ";
236         for(let i = 0; i < space - 1; ++i){
237             s += "\"";
238             s += nodo.right.point.vectorialSpace[i];
239             s += ",";
240         }
```

```
241         s += nodo.right.point.vectorialSpace[space -1];
242         s += "\";\n";
243         cola.push(nodo.right);
244     }
245     s += "}"
246     return s;
247 }
248
249 function buildKDTree(points, depth = 0) {
250     if(points.length === 0) {
251         return;
252     } else {
253         mergeSort(points, 0, points.length - 1, depth % points[0].vectorialSpace.length);
254         let median = Math.floor(points.length / 2);
255         let root = new Node(points[median], depth % points[0].vectorialSpace.length);
256         points.splice(median, 1);
257         let leftBranch = points.slice(0, median);
258         let rightBranch = points.slice(median, points.length);
259
260         root.left = buildKDTree(leftBranch, depth + 1);
261         root.right = buildKDTree(rightBranch, depth + 1);
262
263         return root;
264     }
265
266 }
267
268 function mergeSort(points, left, right, dim) {
269     let mid = Math.floor((left + right) / 2);
270
271     if(left < right) {
272         mergeSort(points, left, mid, dim);
273         mergeSort(points, mid + 1, right, dim);
274         merge(points, left, mid, right, dim);
275     }
276 }
277
278 function merge(points, left, mid, right, dim) {
279     let temp = [];
280     let i = left;
281     let j = mid + 1;
282
283     while(i <= mid && j <= right) {
284         if(points[i].vectorialSpace[dim] <= points[j].vectorialSpace[dim])
285             temp.push(points[i++]);
286         else
287             temp.push(points[j++]);
288     }
289
290     while(i <= mid)
291         temp.push(points[i++]);
292     while(j <= right)
```

```
293         temp.push(points[j++]);
294 
295     for(let i = left, j = 0; i <= right; ++i, ++j)
296         points[i] = temp[j];
297 }
```

Código 5: sketch.js

```
1  function setup(){
2      var width = 250;
3      var height = 200;
4  
5      var canvas = createCanvas(width, height);
6      canvas.parent('sketch_holder');
7      background(0);
8      for( var x = 0; x < width; x += width / 10){
9          for( var y = 0; y < height; y += height /5){
10             stroke(125, 125, 125);
11             strokeWeight(1);
12             line(x, 0, x, height);
13             line(0, y, width, y);
14         }
15     }
16 
17     var data = [];
18     var point = [100,100];
19 
20     //random data
21 
22     for(let i = 0; i < 1000; i++){
23         var x = Math.floor(Math.random() * width);
24         var y = Math.floor(Math.random() * height);
25         let newPoint = new N_Point([x,y]);
26         data.push(newPoint);
27     }
28 
29     //build KD-tree with data
30 
31     var dataChange = data.slice(); //variable temporal porque la siguiente linea modifica data !!!!!
32     var root = buildKDTree(dataChange);
33 
34     console.log(root);
35 
36     // closest point
37     var closestPoint1 = closest_point_brute_force(data, point);
38     var closestPoint2 = naive_closest_point(root, point);
39     var closestPoint3 = closest_point(root, point);
40 
41     console.log("CLOSEST POINT BRUTE FORCE : "+ closestPoint1);
42     console.log("NAIVE CLOSEST POINT NAIVE : "+ closestPoint2);
43     console.log("CLOSEST POINT: " + closestPoint3);
44 
```

```
45      let knnPoints = range_query_rec(data,point, 20, null);
46      console.log(knnPoints);
47
48      //plot points
49      for(let i=0;i<data.length;i++){
50          x = data[i].vectorialSpace[0];
51          y = data[i].vectorialSpace[1];
52           fill(255, 255, 255);
53           circle(x, height - y, 4);
54          textSize(8);
55      }
56      x = closestPoint2[0];
57      y = closestPoint2[1];
58
59
60      for(let i = 0; i < knnPoints.length; ++i) {
61          x = knnPoints[i][0];
62          y = knnPoints[i][1];
63           fill(57, 255, 20);
64           circle(x, height - y, 4);
65          textSize(8);
66      }
67
68      var x = point[0];
69      var y = point[1];
70
71       fill(81, 209, 246);
72       circle(x, height - y, 7);
73      textSize(8);
74      text(x + ',' + y, x + 5, height - y);
75
76      // plot graph
77      var graph = generateDot(root);
78
79      var options = {
80          format: 'svg'
81      }
82
83      var image = Viz(graph, options);
84      var graph_holder = document.getElementById('graph_holder');
85
86      graph_holder.innerHTML = image;    // SVG
87
88  }
```

4. Capturas de pantalla

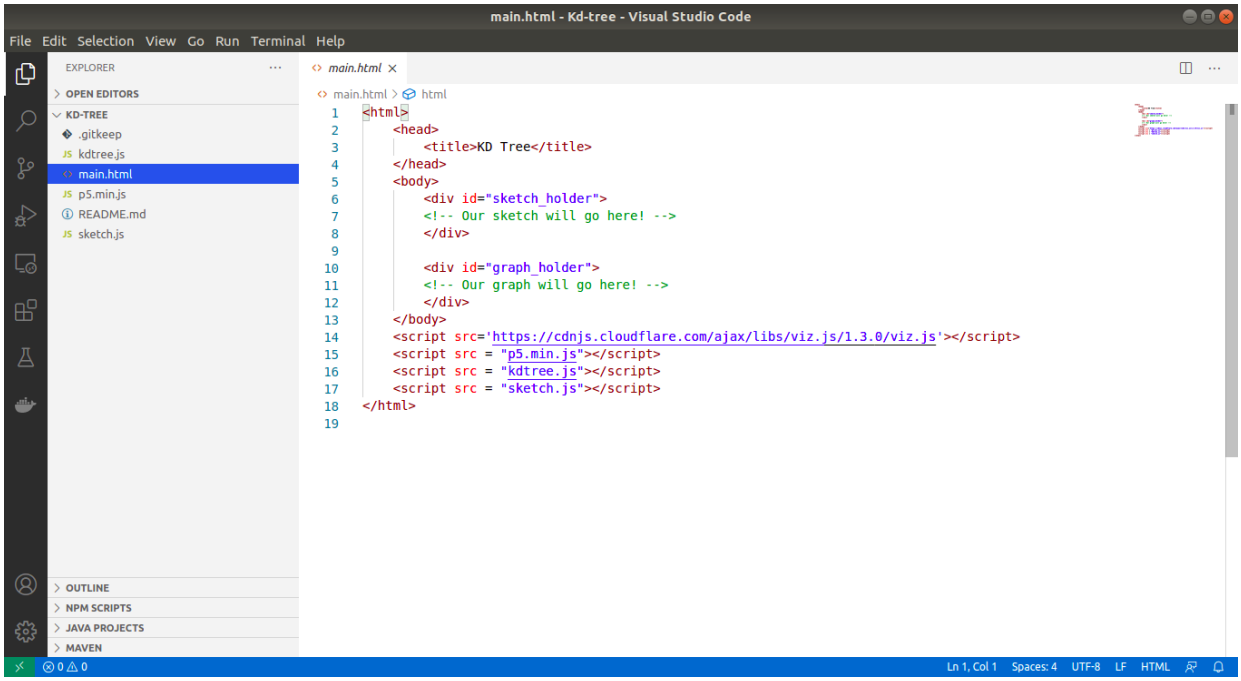

Figura 3: Captura de Pantalla de main.html

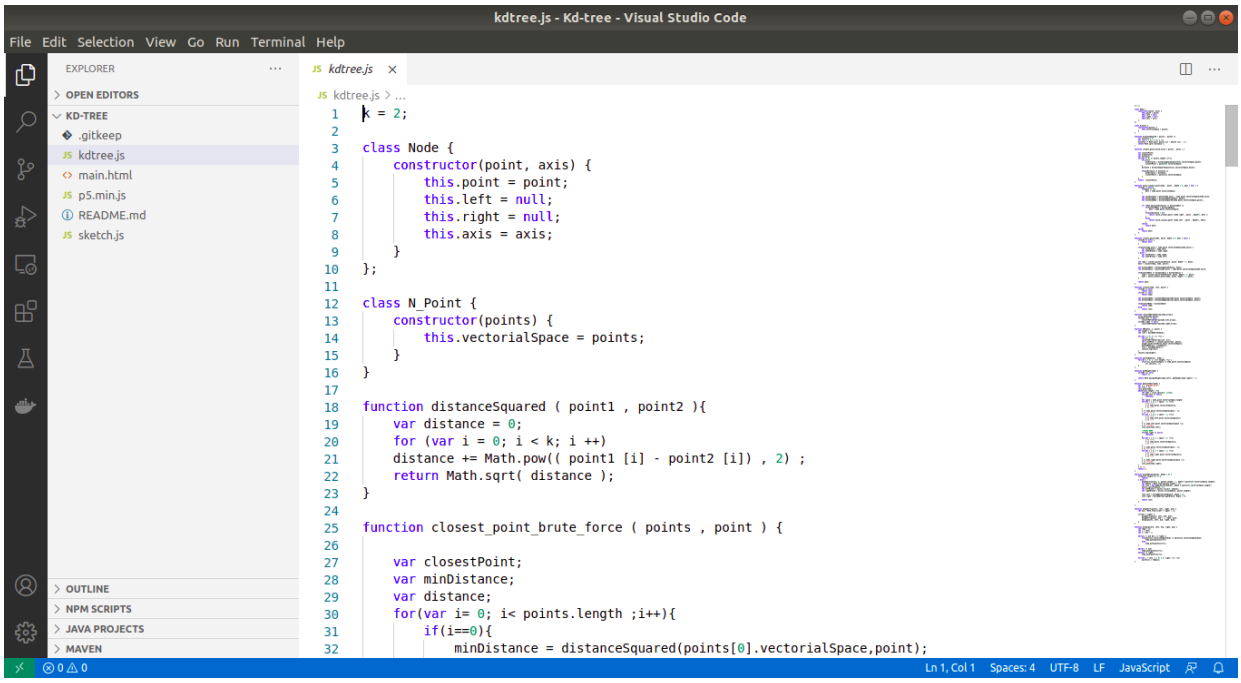

Figura 4: Captura de Pantalla de kdtree.js

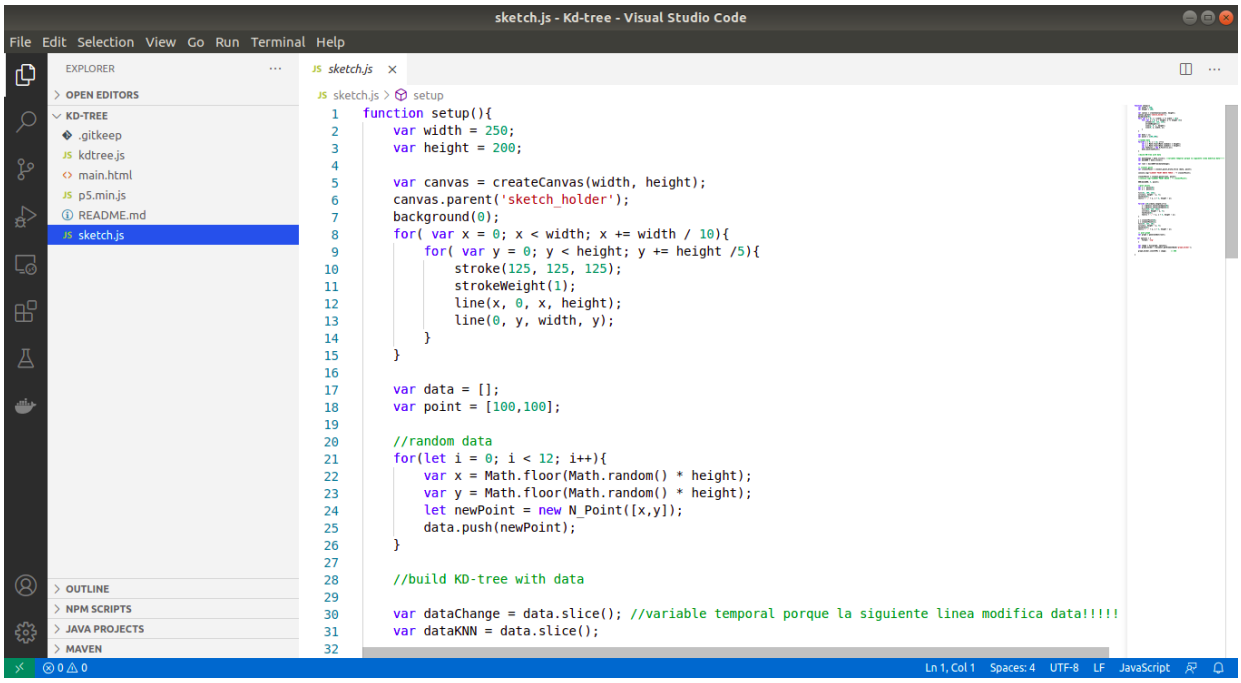

Figura 5: Captura de Pantalla de sketch.js

5. Repositorios

Toda la implementación de la estructura KD-Tree, el código, documento, imágenes y un READ-ME, se encuentran en los siguientes repositorios

- GitHub: https://github.com/Leslym03/EDA-Grupo1/tree/master/KDTree
- GitLab: https://gitlab.com/pimed/kdtree/-/tree/master