

Practica 5 y 6

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: 10 de noviembre de 2020
Arequipa, Perú

Índice de Contenidos

1. Introducción	1
1.1. Presentación y objetivos	1
1.2. Herramientas	1
1.3. Estructura del documento	2
2. Implementación	3
2.1. Nodo	3
2.2. Funciones	3
2.2.1. Get Height	3
2.2.2. Generate Dot	4
2.2.3. Build kdtree	5
2.2.4. Distance Squared	6
2.2.5. Closest Point Brute Force	6
2.2.6. Naive Closest Point	6
2.2.7. Closest Point	7
2.2.8. KNN	8
2.2.9. Closest	8
2.2.10. Convert KD-Tree To Array	9
2.2.11. Delete Node	9
3. Evaluación y Pruebas	10
3.1. Sketch	10
3.2. Datos	11
4. Conclusión	14
Anexo A. Código fuente	15
Anexo B. Capturas de pantalla	22
Anexo C. Repositorios	24

Índice de Figuras

1. KD-Tree en 2 dimensiones	1
2. Herramientas utilizadas	1
3. Visualización del navegador Web	11
4. Visualización con Datos 1	12
5. Visualización con Datos 2	13
B.1. Captura de Pantalla de main.html	22
B.2. Captura de Pantalla de kdtree.js	23
B.3. Captura de Pantalla de sketch.js	23

Índice de Códigos

1.	class Node	3
2.	N_Point	3
3.	getHeight	3
4.	generateDot	4
5.	build kdtree	5
6.	distanceSquared	6
7.	closest_point_brute_force	6
8.	naive_closest_point	7
9.	closest_point	7
10.	KNN	8
11.	closest	8
12.	convertKDTreeToArray	9
13.	deleteNode	9
14.	Sketch	10
15.	Datos 1	11
16.	Datos 2	12
A.1.	main.html	15
A.2.	kdtree.js	15
A.3.	sketch.js	20

1. Introducción

1.1. Presentación y objetivos

Los KD-Tree (árboles de dimensión k) son una herramienta matemática que sirve para dividir o particionar el espacio, organizando los puntos que se encuentran en una construcción geométrica perteneciente a un espacio k -dimensional.

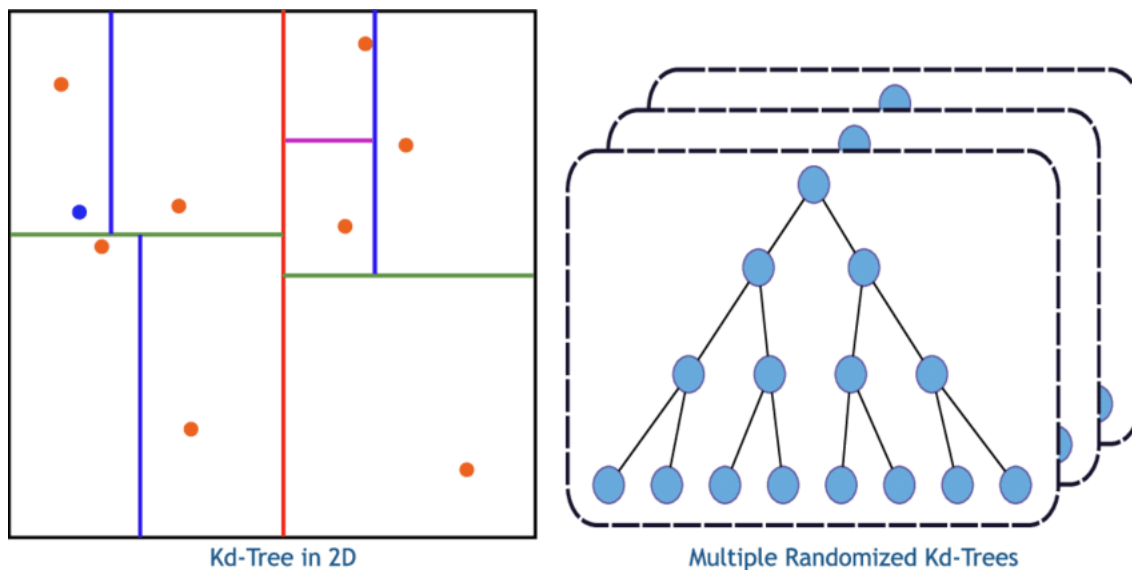


Figura 1: KD-Tree en 2 dimensiones

La presente consiste en una implementación de la estructura KD-Tree con distintos métodos que posee, para ello se utilizaron herramientas y se estructuró el presente documento.

1.2. Herramientas

Para la implementación como lenguaje de programación se utilizó Javascript para la implementación de la estructura y para el procesamiento de la imagen se utilizó la librería p5.js, esto debido a que es muy práctico ya que simplifica en gran medida el proceso de graficar imágenes.

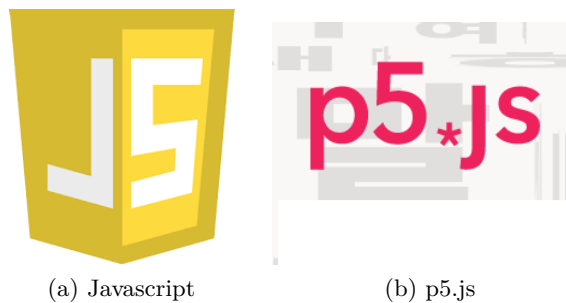


Figura 2: Herramientas utilizadas

1.3. Estructura del documento

El presente documento está dividido en una serie de secciones que conforman el proceso de desarrollo del proyecto, detalladas a continuación:

- **Implementación:** Se detalla el desarrollo de la implementación de todos los métodos necesarios para KD-Tree.
- **Evaluación y Pruebas:** Se detalla los resultados en KD-Tree para la comprobación del correcto funcionamiento mediante una serie de pruebas.
- **Conclusión:** Se detallan las observaciones sobre KD-Tree, en función a la implementación y capacidades de este.

2. Implementación

Se comenzó creando el archivo *main.html*. La captura de pantalla del **Código A.1** se encuentra en la sección Capturas de pantalla como **Figura B.1**.

2.1. Nodo

Seguidamente se creo un archivo *kdtree.js*. La captura de pantalla del **Código A.1** se encuentra en la sección Capturas de pantalla como **Figura B.2**.

Creamos una clase Nodo que albergar los puntos , su constructor tiene como parámetros el punto y el eje del mismo.

Código 1: class Node

```
1 k = 2;
2
3 class Node {
4     constructor(point, axis) {
5         this.point = point;
6         this.left = null;
7         this.right = null;
8         this.axis = axis;
9     }
10 };
```

También se creo una clase N_Points que contiene un array para albergar las diferentes dimensiones del punto, en este caso utilizaremos 2 dimensiones dentro del KD-Tree

Código 2: N_Point

```
1 class N_Point {
2     constructor(points) {
3         this.vectorialSpace = points;
4     }
5 }
```

2.2. Funciones

2.2.1. Get Height

Retorna la altura de un nodo, recursivamente tomando como parámetro el nodo a hallar su altura, la lógica es la misma que la de un árbol binario.

Código 3: getHeight

```
1 function getHeight(node) {
2     if(node === null)
3         return 0;
4     return Math.max(getHeight(node.left), getHeight(node.right)) + 1;
5 }
```

2.2.2. Generate Dot

Genera un árbol en formato dot a partir de enviarle como parámetro el nodo raíz, para luego poder ser graficado en el archivo *skech.js*

Código 4: generateDot

```
1 function generateDot(node) {
2     var s = "digraph G{\n\"";
3     var cola = [];
4     cola.push(node);
5     while(cola.length > 0){
6         let nodo = (cola.splice(0, 1)) [0];
7         if(nodo.left == null){
8             continue;
9         }
10        let space = nodo.point.vectorialSpace.length;
11        for(let i = 0; i < space - 1; ++i){
12            s += nodo.point.vectorialSpace[i];
13            s += ",";
14        }
15        s += nodo.point.vectorialSpace[space - 1];
16        s += "\" -> ";
17        for(let i = 0; i < space - 1; ++i){
18            s += nodo.left.point.vectorialSpace[i];
19            s += ",";
20        }
21        s += nodo.left.point.vectorialSpace[space - 1];
22        s += "\";\n";
23        cola.push(nodo.left);
24
25        //RIGHT NODE
26        if(nodo.right == null){
27            continue;
28        }
29        for(let i = 0; i < space - 1; ++i){
30            s += nodo.point.vectorialSpace[i];
31            s += ",";
32        }
33        s += nodo.point.vectorialSpace[space - 1];
34        s += "\" -> ";
35        for(let i = 0; i < space - 1; ++i){
36            s += nodo.right.point.vectorialSpace[i];
37            s += ",";
38        }
39        s += nodo.right.point.vectorialSpace[space - 1];
40        s += "\";\n";
41        cola.push(nodo.right);
42    }
43    s += "}";
44    console.log(s);
45 }
```

2.2.3. Build kdtree

Construye el KD-Tree y retorna el nodo raiz, para ello tambien se implemento el algoritmo de ordenamiento mergesort para los puntos del KD-Tree.

Código 5: build kdtree

```
1 function buildKDTree(points, depth = 0) {
2   if(points.length === 0) {
3     return;
4   } else {
5     mergeSort(points, 0, points.length - 1, depth % points[0].vectorialSpace.length);
6     let median = Math.floor(points.length / 2);
7     let root = new Node(points[median], depth % points[0].vectorialSpace.length);
8     points.splice(median, 1);
9     let leftBranch = points.slice(0, median);
10    let rightBranch = points.slice(median, points.length);
11
12    root.left = buildKDTree(leftBranch, depth + 1);
13    root.right = buildKDTree(rightBranch, depth + 1);
14
15    return root;
16  }
17 }
18
19
20 function mergeSort(points, left, right, dim) {
21   let mid = Math.floor((left + right) / 2);
22
23   if(left < right) {
24     mergeSort(points, left, mid, dim);
25     mergeSort(points, mid + 1, right, dim);
26     merge(points, left, mid, right, dim);
27   }
28 }
29
30 function merge(points, left, mid, right, dim) {
31   let temp = [];
32   let i = left;
33   let j = mid + 1;
34
35   while(i <= mid && j <= right) {
36     if(points[i].vectorialSpace[dim] <= points[j].vectorialSpace[dim])
37       temp.push(points[i++]);
38     else
39       temp.push(points[j++]);
40   }
41
42   while(i <= mid)
43     temp.push(points[i++]);
44   while(j <= right)
45     temp.push(points[j++]);
46 }
```



```

47   for(let i = left, j = 0; i <= right; ++i, ++j)
48       points[i] = temp[j];
49 }

```

2.2.4. Distance Squared

Encuentra la distancia entre dos puntos según la dimensión de estos.

Código 6: distanceSquared

```

1 function distanceSquared(point1 , point2){
2     var distance = 0;
3     for(var i = 0; i < k; i++)
4         distance += Math.pow((point1[i] - point2[i]), 2);
5     return Math.sqrt(distance);
6 }

```

2.2.5. Closest Point Brute Force

La función nos retorna el punto más cercano dentro de la estructura KD-Tree, recibe como parámetros un array de puntos y un punto que a partir de este se buscara el mas cercano.

Código 7: closest_point_brute_force

```

1 function closest_point_brute_force ( points , point ) {
2
3     var closestPoint ;
4     var minDistance;
5     var distance;
6     for(var i= 0; i< points.length ;i++){
7         if(i==0){
8             minDistance = distanceSquared(points[0].vectorialSpace,point);
9             closestPoint = points[0]. vectorialSpace;
10        }
11        distance = distanceSquared(points[i]. vectorialSpace,point);
12
13        if(minDistance >= distance ){
14            minDistance = distance;
15            closestPoint = points[i]. vectorialSpace;
16        }
17    }
18    return closestPoint ;
19 }

```

2.2.6. Naive Closest Point

La función recibe como parámetros el nodo raíz de un Kd-tree , el punto para que el queremos hallar el punto más cercano, la profundidad (por defecto 0) y best (alberga el mejor punto) este comienza inicialmente como null. La función retorna el punto más cercano a point.

Código 8: naive_closest_point

```

1
2 function naive_closest_point(node , point , depth = 0, best = null ) {
3     if (node !== null){
4         if (depth == 0){
5             best = node.point.vectorialSpace;
6         }
7
8         var axisDistance = point[node.axis] - node.point.vectorialSpace[node.axis];
9         let distanceBest = distanceSquared(best, point);
10        let distanceNode = distanceSquared(node.point.vectorialSpace,point);
11
12
13        if ( Math.abs(axisDistance) <= distanceBest ){
14            if (distanceBest > distanceNode){
15                best = node.point.vectorialSpace;
16            }
17            if (axisDistance > 0){
18                return naive_closest_point (node.right , point , depth+1, best );
19            }
20            else{
21                return naive_closest_point (node.left , point , depth+1, best);
22            }
23        } else{
24            return best;
25        }
26    } else{
27        return best;
28    }
29 }

```

2.2.7. Closest Point

La función tiene como parámetros el nodo raíz de un Kd-tree previamente antes armado , el punto para el que buscaremos el punto más cercano y la profundidad que inicialmente esta como 0. La función retorna el punto más cercano al punto de entrada dado(point).

Código 9: closest_point

```

1 function closest_point(node, point, depth = 0) {
2     if (node == null) {
3         return null;
4     }
5
6     if (point[node.axis] < node.point.vectorialSpace[node.axis]) {
7         var nextBranch = node.left;
8         var otherBranch = node.right;
9     } else {
10        var nextBranch = node.right;
11        var otherBranch = node.left;
12    }

```

```
13
14     var temp = closest_point(nextBranch, point, depth + 1);
15     var best = closest(temp, node, point);
16
17     var distanceBest = distanceSquared(point, best.point.vectorialSpace);
18     var distanceAxis = Math.abs(point[node.axis] - node.point.vectorialSpace[node.axis]);
19
20     if(distanceAxis <= distanceBest) {
21         temp = closest_point(otherBranch, point, depth + 1);
22         best = closest(temp, best, point);
23     }
24
25     return best;
26 }
```

2.2.8. KNN

La función tiene como parámetros data que es un array de puntos , n la cantidad de puntos cercanos que queremos hallar y point que es el punto al que se sacaremos los n puntos cercanos. La función retorna un array con los n puntos mas cercanos a point.

Código 10: KNN

```
1
2 function KNN(data, n, point) {
3     let neight = [];
4     let root = buildKDTree(data);
5
6     for(let i = 0; i < n; ++i) {
7         let arr = [];
8         convertKDTreeToArray(root, arr);
9         let closePoint = closest_point(root, point);
10        neight.push(closePoint.point.vectorialSpace);
11        deleteNode(arr, closePoint);
12        root = buildKDTree(arr);
13    }
14    return neight;
15 }
```

2.2.9. Closest

Calcula la distancia de un punto hacia la raíz encontrando el mayor y menor entre ambos puntos.

Código 11: closest

```
1 function closest (node, root, point) {
2     if (node == null)
3         return root;
4     if (root == null)
5         return node;
6 }
```

```
7   let distanceNode = distanceSquared(node.point.vectorialSpace, point);
8   let distanceRoot = distanceSquared(root.point.vectorialSpace, point);
9
10  if(distanceNode < distanceRoot)
11    return node;
12  else
13    return root;
14 }
```

2.2.10. Convert KD-Tree To Array

La función recibe como entrada el nodo raíz de un KD-Tree ya armado y un array . La función retorna array con los puntos del KD-Tree

Código 12: convertKDTreeToArray

```
1 function convertKDTreeToArray(node,array){
2   array.push(node.point);
3   if (node.left !== null)
4     convertKDTreeToArray(node.left,array);
5   if (node.right !== null)
6     convertKDTreeToArray(node.right,array);
7 }
```

2.2.11. Delete Node

Elimina un nodo del espacio vectorial en el que se encuentra a partir de los parámetros de su arreglo y el mismo nodo.

Código 13: deleteNode

```
1 function deleteNode(arr, node) {
2   for(let i = 0; i < arr.length; ++i) {
3     if(arr[i].vectorialSpace == node.point.vectorialSpace)
4       arr.splice(i, 1);
5   }
6 }
```

3. Evaluación y Pruebas

La captura de pantalla del **Código A.3** se encuentra en la sección Capturas de pantalla como **Figura B.3**.

3.1. Sketch

Código 14: Sketch

```
1 function setup(){
2   var width = 250;
3   var height = 200;
4   createCanvas(width, height);
5
6   background(0);
7   for( var x = 0; x < width; x += width / 10){
8     for( var y = 0; y < height; y += height / 5){
9       stroke(125, 125, 125);
10      strokeWeight(1);
11      line(x, 0, x, height);
12      line(0, y, width, y);
13    }
14  }
15
16  var data = [];
17  for(let i = 0; i < 12; i++){
18    var x = Math.floor(Math.random() * height);
19    var y = Math.floor(Math.random() * height);
20    let newPoint = new N__Point([x,y]);
21    data.push(newPoint);
22
23    fill (255, 255, 255);
24    circle (x, height - y, 7);
25    textSize(8);
26    text(x + ',' + y, x + 5, height - y);
27  }
28
29  var root = buildKDTree(data);
30  generateDot(root);
31  console.log(root);
32 }
```

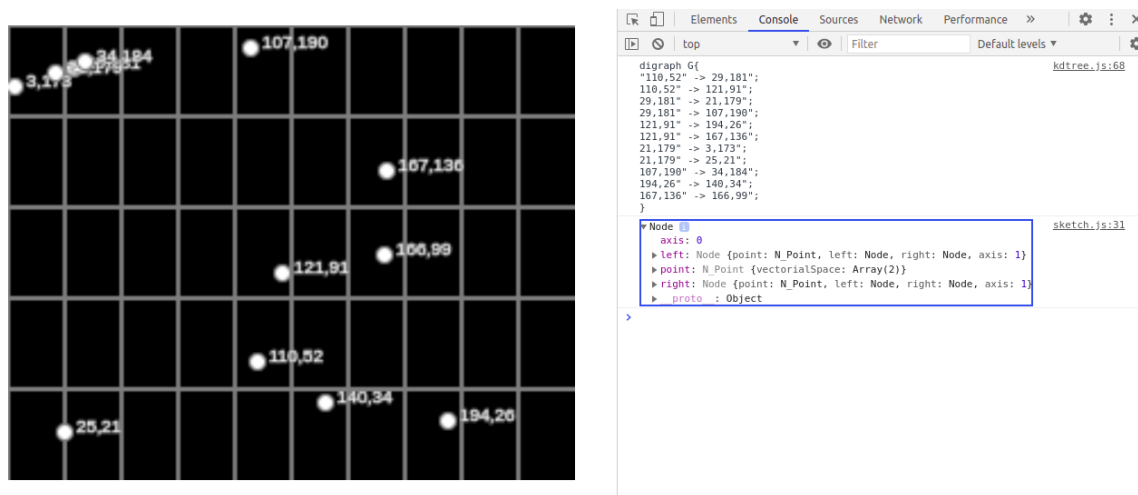


Figura 3: Visualización del navegador Web

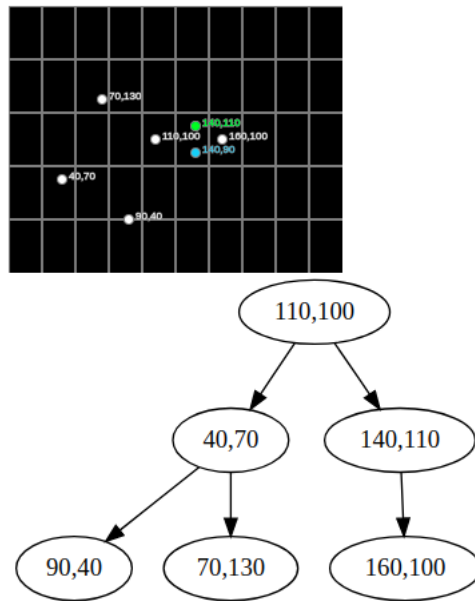
Se puede ver en la **Figura 3**, en el cual los puntos están en el árbol KD-Tree creándose sus hijos de cada uno en el espacio. También se muestra en la consola digraph G en el formato que se pidió.

3.2. Datos

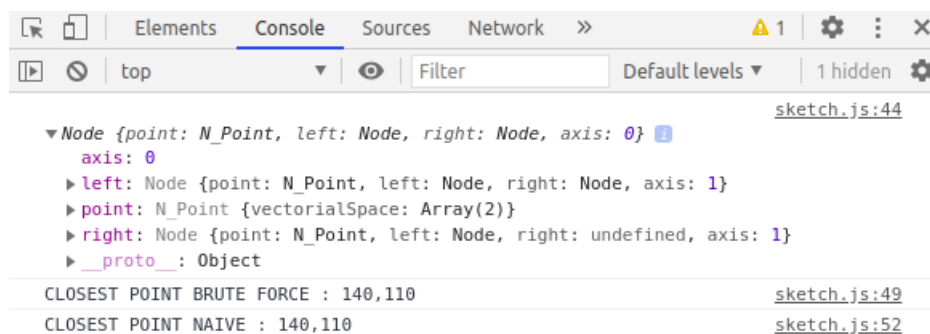
Evalúe el resultado de las dos funciones implementadas anteriormente con este conjunto de datos:

Código 15: Datos 1

```
1 var data = [
2   [40 ,70] ,
3   [70 ,130] ,
4   [90 ,40] ,
5   [110 , 100] ,
6   [140 ,110] ,
7   [160 , 100]
8 ];
9 var point = [140 ,90]; // query
```



(a) Web



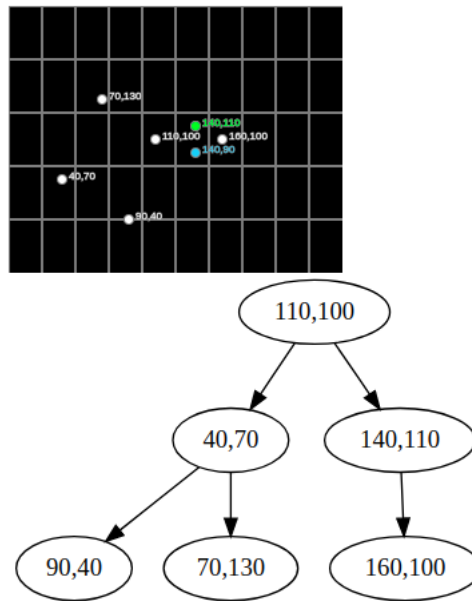
(b) Consola

Figura 4: Visualización con Datos 1

Código 16: Datos 2

```

1 var data = [
2   [40 ,70] ,
3   [70 ,130] ,
4   [90 ,40] ,
5   [110 , 100] ,
6   [140 ,110] ,
7   [160 , 100] ,
8   [150 , 30]
9 ];
10 var point = [140 ,90]; // query
  
```



(a) Web

```
▼ Node {point: N_Point, left: Node, right: Node, axis: 0} ⓘ  
  axis: 0  
  ▶ left: Node {point: N_Point, left: Node, right: Node, axis: 1}  
  ▶ point: N_Point {vectorialSpace: Array(2)}  
  ▶ right: Node {point: N_Point, left: Node, right: undefined, axis: 1}  
  ▶ __proto__: Object  
  
CLOSEST POINT BRUTE FORCE : 140,110  
CLOSEST POINT NAIVE : 140,110
```

(b) Consola

Figura 5: Visualización con Datos 2

4. Conclusión

- Los kd-trees son una generalización de los árboles binarios de búsqueda que permiten manejar elementos con claves de más de una dimensión.
- Un kd-tree de tamaño n sea aleatorio, implica que el coste en el caso promedio para sus operaciones, no siempre un kd-tree conserva la aleatoriedad con la que fue construido.
- Los kd-trees son frecuentemente utilizados en bases de datos para satisfacer consultas que incluyan valores de varios campos.

Anexo A. Código fuente

Código A.1: main.html

```
1 <html>
2   <head>
3     <title>KD Tree</title>
4     <script src = "p5.min.js"></script>
5     <script src = "kdtree.js"></script>
6     <script src = "sketch.js"></script>
7   </head>
8   <body>
9
10  </body>
11 </html>
```

Código A.2: kdtree.js

```
1 k = 2;
2
3 class Node {
4   constructor(point, axis) {
5     this.point = point;
6     this.left = null;
7     this.right = null;
8     this.axis = axis;
9   }
10 };
11
12 class N_Point {
13   constructor(points) {
14     this.vectorialSpace = points;
15   }
16 }
17
18 function distanceSquared ( point1 , point2 ){
19   var distance = 0;
20   for (var i = 0; i < k; i ++){
21     distance += Math.pow(( point1 [i] - point2 [i]) , 2) ;
22   }
23   return Math.sqrt( distance );
24 }
25
26 function closest_point_brute_force ( points , point ) {
27   var closestPoint ;
28   var minDistance;
29   var distance;
30   for(var i= 0; i< points.length ;i++){
31     if(i==0){
32       minDistance = distanceSquared(points[0].vectorialSpace,point);
33       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 function closest_point(node, point, depth = 0) {
75     if(node == null) {
76         return null;
77     }
78
79     if(point[node.axis] < node.point.vectorialSpace[node.axis]) {
80         var nextBranch = node.left;
81         var otherBranch = node.right;
82     } else {
83         var nextBranch = node.right;
84         var otherBranch = node.left;
85     }
```

```
86
87     var temp = closest_point(nextBranch, point, depth + 1);
88     var best = closest(temp, node, point);
89
90     var distanceBest = distanceSquared(point, best.point.vectorialSpace);
91     var distanceAxis = Math.abs(point[node.axis] - node.point.vectorialSpace[node.axis]);
92
93     if(distanceAxis <= distanceBest) {
94         temp = closest_point(otherBranch, point, depth + 1);
95         best = closest(temp, best, point);
96     }
97
98     return best;
99 }
100
101 function closest(node, root, point) {
102     if(node == null)
103         return root;
104     if(root == null)
105         return node;
106
107     let distanceNode = distanceSquared(node.point.vectorialSpace, point);
108     let distanceRoot = distanceSquared(root.point.vectorialSpace, point);
109
110     if(distanceNode < distanceRoot)
111         return node;
112     else
113         return root;
114 }
115
116 function convertKDTreeToArray(node,array){
117     array.push(node.point);
118     if(node.left != null)
119         convertKDTreeToArray(node.left,array);
120     if(node.right != null)
121         convertKDTreeToArray(node.right,array);
122 }
123
124 function KNN(data, n, point) {
125     let neight = [];
126     let root = buildKDTree(data);
127
128     for(let i = 0; i < n; ++i) {
129         let arr = [];
130         convertKDTreeToArray(root, arr);
131         let closePoint = closest_point(root, point);
132         neight.push(closePoint.point.vectorialSpace);
133         deleteNode(arr, closePoint);
134         root = buildKDTree(arr);
135     }
136     return neight;
137 }
```

```
138
139 function deleteNode(arr, node) {
140     for(let i = 0; i < arr.length; ++i) {
141         if(arr[i].vectorialSpace == node.point.vectorialSpace)
142             arr.splice(i, 1);
143     }
144 }
145
146 function getHeight(node) {
147     if(node === null)
148         return 0;
149
150     return Math.max(getHeight(node.left), getHeight(node.right)) + 1;
151 }
152
153 function generateDot(node) {
154     var s = "digraph G{\n";
155     var cola = [];
156     cola.push(node);
157     while(cola.length > 0){
158         let nodo = (cola.splice(0, 1)) [0];
159         if(nodo.left == null){
160             continue;
161         }
162         let space = nodo.point.vectorialSpace.length;
163         for(let i = 0; i < space - 1; ++i){
164             s += "\n";
165             s += nodo.point.vectorialSpace[i];
166             s += ",";
167         }
168         s += nodo.point.vectorialSpace[space - 1];
169         s += "\n -> ";
170         for(let i = 0; i < space - 1; ++i){
171             s += "\n";
172             s += nodo.left.point.vectorialSpace[i];
173             s += ",";
174         }
175         s += nodo.left.point.vectorialSpace[space - 1];
176         s += "\n;\n";
177         cola.push(nodo.left);
178
179         //RIGHT NODE
180         if(nodo.right == null){
181             continue;
182         }
183         for(let i = 0; i < space - 1; ++i){
184             s += "\n";
185             s += nodo.point.vectorialSpace[i];
186             s += ",";
187         }
188         s += nodo.point.vectorialSpace[space - 1];
189         s += "\n -> ";

```

```
190     for(let i = 0; i < space - 1; ++i){
191         s += "\"";
192         s += nodo.right.point.vectorialSpace[i];
193         s += ",";
194     }
195     s += nodo.right.point.vectorialSpace[space - 1];
196     s += "\";\n";
197     cola.push(nodo.right);
198 }
199 s += "}"
200 return s;
201 }
202
203 function buildKDTree(points, depth = 0) {
204     if(points.length === 0) {
205         return;
206     } else {
207         mergeSort(points, 0, points.length - 1, depth % points[0].vectorialSpace.length);
208         let median = Math.floor(points.length / 2);
209         let root = new Node(points[median], depth % points[0].vectorialSpace.length);
210         points.splice(median, 1);
211         let leftBranch = points.slice(0, median);
212         let rightBranch = points.slice(median, points.length);
213
214         root.left = buildKDTree(leftBranch, depth + 1);
215         root.right = buildKDTree(rightBranch, depth + 1);
216
217         return root;
218     }
219 }
220 }
221
222 function mergeSort(points, left, right, dim) {
223     let mid = Math.floor((left + right) / 2);
224
225     if(left < right) {
226         mergeSort(points, left, mid, dim);
227         mergeSort(points, mid + 1, right, dim);
228         merge(points, left, mid, right, dim);
229     }
230 }
231
232 function merge(points, left, mid, right, dim) {
233     let temp = [];
234     let i = left;
235     let j = mid + 1;
236
237     while(i <= mid && j <= right) {
238         if(points[i].vectorialSpace[dim] <= points[j].vectorialSpace[dim])
239             temp.push(points[i++]);
240         else
241             temp.push(points[j++]);
```

```

242 }
243
244 while(i <= mid)
245     temp.push(points[i++]);
246 while(j <= right)
247     temp.push(points[j++]);
248
249 for(let i = left, j = 0; i <= right; ++i, ++j)
250     points[i] = temp[j];
251 }

```

Código A.3: 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     for(let i = 0; i < 12; i++){
22         var x = Math.floor(Math.random() * height);
23         var y = Math.floor(Math.random() * height);
24         let newPoint = new N_Point([x,y]);
25         data.push(newPoint);
26     }
27
28     //build KD-tree with data
29
30     var dataChange = data.slice(); //variable temporal porque la siguiente linea modifica data !!!!!
31     var dataKNN = data.slice();
32
33     var root = buildKDTree(dataChange);
34
35     // closest point
36     var closestPoint = closest_point_brute_force (data, point);
37
38     console.log("CLOSEST POINT BRUTE FORCE : "+ closestPoint);
39

```

```
40  closestPoint = closest_point(root, point);
41  //console.log("CLOSEST POINT NAIVE : "+ closestPoint);
42
43  KNN(dataKNN, 5, point);
44
45  //plot points
46  var x = point[0];
47  var y = point[1];
48
49  fill (81, 209, 246);
50  circle (x, height - y, 7);
51  textSize(8);
52  text(x + ',' + y, x + 5, height - y);
53
54
55  for(let i=0;i<data.length;i++){
56    x = data[i]. vectorialSpace [0];
57    y = data[i]. vectorialSpace [1];
58    fill (255, 255, 255);
59    circle (x, height - y, 7);
60    textSize(8);
61    text(x + ',' + y, x + 5, height - y);
62  }
63
64  x = closestPoint [0];
65  y = closestPoint [1];
66  fill (57, 255, 20);
67  circle (x, height - y, 7);
68  textSize(8);
69  text(x + ',' + y, x + 5, height - y);
70
71  // plot graph
72  var graph = generateDot(root);
73
74  var options = {
75    format: 'svg'
76  }
77
78  var image = Viz(graph, options);
79  var graph_holder = document.getElementById('graph_holder');
80
81  graph_holder.innerHTML = image;  // SVG
82
83 }
```


Anexo B. Capturas de pantalla

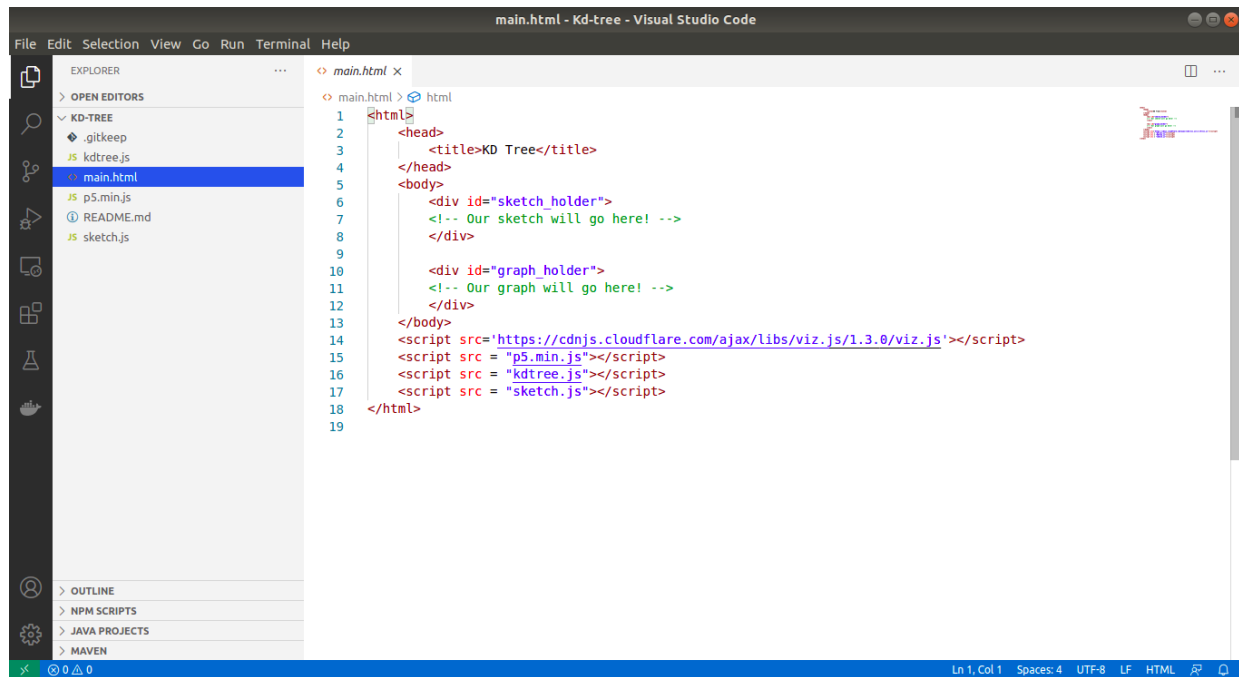
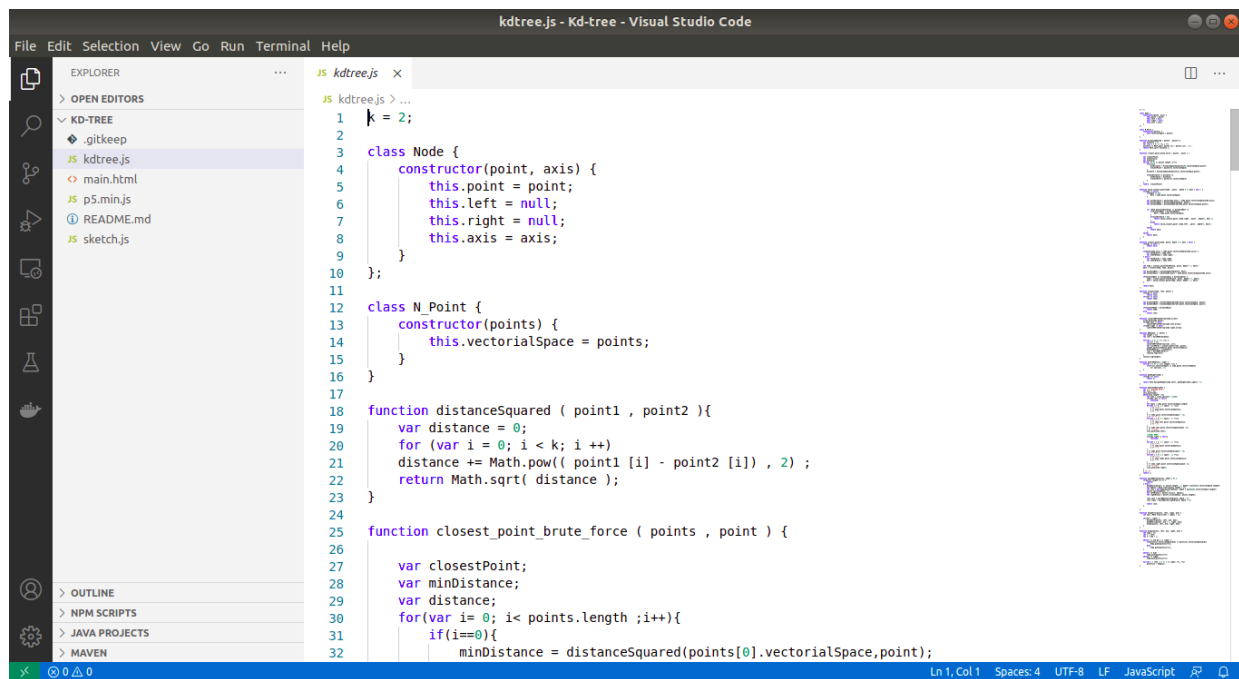
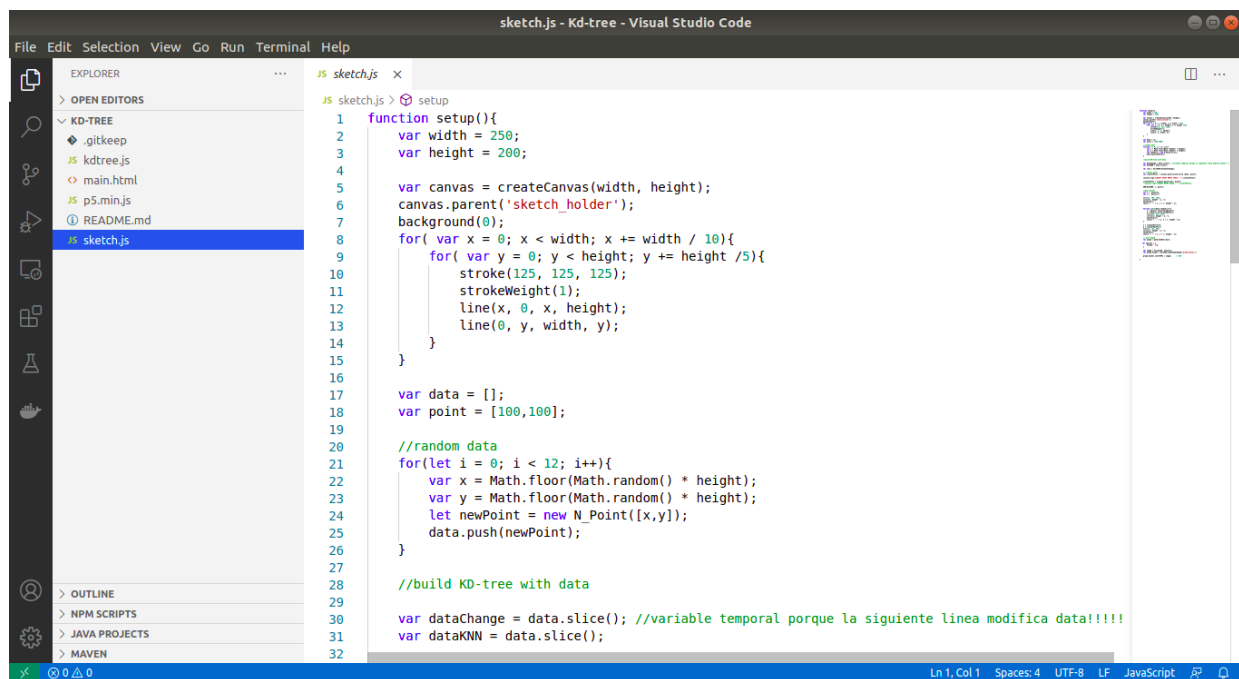


Figura B.1: Captura de Pantalla de main.html



```
1 k = 2;
2
3 class Node {
4   constructor(point, axis) {
5     this.point = point;
6     this.left = null;
7     this.right = null;
8     this.axis = axis;
9   }
10 }
11
12 class N_Point {
13   constructor(points) {
14     this.vectorialSpace = points;
15   }
16 }
17
18 function distanceSquared ( point1 , point2 ){
19   var distance = 0;
20   for (var i = 0; i < k; i ++){
21     distance += Math.pow(( point1 [i] - point2 [i]) , 2) ;
22   }
23   return Math.sqrt( distance );
24 }
25
26 function closest_point_brute_force ( points , point ) {
27   var closestPoint;
28   var minDistance;
29   var distance;
30   for(var i= 0; i< points.length ;i++){
31     if(i==0){
32       minDistance = distanceSquared(points[0].vectorialSpace,point);
```

Figura B.2: Captura de Pantalla de kdtree.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   for(let i = 0; i < 12; i++){
22     var x = Math.floor(Math.random() * height);
23     var y = Math.floor(Math.random() * height);
24     let newPoint = new N_Point([x,y]);
25     data.push(newPoint);
26   }
27
28   //build KD-tree with data
29
30   var dataChange = data.slice(); //variable temporal porque la siguiente linea modifica data!!!!
31   var dataKNN = data.slice();
32 }
```

Figura B.3: Captura de Pantalla de sketch.js

Anexo C. Repositorios

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

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