Práctica 04

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

PRÁCTICA	TEMA	DURACIÓN
04	Kdtree	3 horas

1. Datos de los estudiantes

- Grupo: V
- Integrantes:
  - Angel Yvan Choquehuanca Peraltilla
  - Estefany Pilar Huamán Colque
  - Eduardo Diaz Huayhuas
  - Gustavo Raul Manrique Fernandez

2. Marco Teorico

2.1. Concepto

El árbol k -d es un árbol binario en el que cada nodo es un punto k -dimensional. Se puede pensar que cada nodo que no es una hoja genera implícitamente un hiperplano de división que divide el espacio en dos partes, conocidas como medios espacios . Los puntos a la izquierda de este hiperplano están representados por el subárbol izquierdo de ese nodo y los puntos a la derecha del hiperplano están representados por el subárbol derecho. La dirección del hiperplano se elige de la siguiente manera: cada nodo en el árbol está asociado con uno de los kdimensiones, con el hiperplano perpendicular al eje de esa dimensión. Entonces, por ejemplo, si para una división particular se elige el eje "x", todos los puntos en el subárbol con un valor "x"más pequeño que el nodo aparecerán en el subárbol izquierdo y todos los puntos con un valor "x"más grande aparecerán. estar en el subárbol derecho. En tal caso, el hiperplano estaría fijado por el valor x del punto, y su normal sería el eje x unitario.

2.2. Particionado

Un árbol kd emplea sólo planos perpendiculares a uno de los ejes del sistema de coordenadas. Todos los nodos de un árbol kd, desde el nodo raíz hasta los nodos hoja, almacenan un punto. Cada plano debe pasar a través de uno de los puntos del árbol kd.

2.3. Repositorio

Github: https://github.com/AngelYvan/mcc-project4-group5

3. Metodologia y Desarrollo

Se realizará una app que permita el desarrollo del algoritmo Kdtree suiendo las tareas preestablecidas por la practica 04.

Tarea 1: Cree un archivo main.html.

Tarea 2: Cree un archivo kdtree.js.

```
function getHeight(node) {
    if(node == null)
        return 0;
    else
        return 1 +(Math.max(getHeight(node.left),getHeight(node.right)));
}
function generate_dot(node) {
    var cad="";
        if(node==null)
                return "";

        if(node.left!=null)
        {
                cad=cad+'"'+node.point.toString()+"\"";
                cad=cad+" -> "+'"'+node.left.point.toString()+'"'+";"+"\n";
        }
        if(node.right!=null)
        {
                cad=cad+"\""+node.point.toString()+"\"";
                cad=cad+" -> "+'"'+node.right.point.toString()+'"'+";"+"\n";
        }
        return cad+generate_dot(node.left)+generate_dot(node.right);
}
function build_kdtree(points, depth = 0) {
    var n = points.length;
    var axis = depth % k;


    if (n <= 0) {
        return null;
    }
    if (n == 1) {
        return new Node(points[0], axis)
    }

    var median = Math.floor(points.length / 2);

    // sort by the axis
    points.sort(function (a, b) {
        return a[axis] - b[axis];
    });
    //console.log(points);

    var left = points.slice(0, median);
    var right = points.slice(median + 1);

    //console.log(right);

    var node = new Node(points[median].slice(0, k), axis);
```

```
    node.left = build_kdtree(left, depth + 1);
    node.right = build_kdtree(right, depth + 1);

    return node;
}
```

Tarea 3: Cree un archivo sketch.js y evalue sus resultados.

```
function setup() {
    var width = 250;
    var height = 200;
    let kdTreeCanvas = createCanvas(width, height);
    kdTreeCanvas.parent("KdTreeCanvas");
    background(0);
    for (var x = 0; x < width; x += width / 10) {
        for (var y = 0; y < height; y += height / 5) {
            stroke(125, 125, 125);
            strokeWeight(1);
            line(x, 0, x, height);
            line(0, y, width, y);
        }
    }
    var data = [];
    for (let i = 0; i < 12; i++) {
        var x = Math.floor(Math.random() * height);
        var y = Math.floor(Math.random() * height);
        data.push([x, y]);
        fill(255, 255, 255);
        circle(x, height - y, 7); // 200 -y para q se dibuje apropiadamente
        textSize(8);
        text(x + ',' + y, x + 5, height - y);// 200 -y para q se dibuje
            apropiadamente
    }

    var root = build_kdtree(data);
    console.log(root);
}
```

Tarea 4: Implemente la funcion closest point brute force y naive closest point.

```
function closest_point_brute_force(points, point) {
    var nearPoints = points[0];
        var DistanceMin = distanceSquared(points[0], point);
        for (var i = 1; i < points.length; i++) {
                var t = distanceSquared(points[i], point);
                if (DistanceMin > t) {
                        nearPoints = points[i];
                        DistanceMin = t;
                }
        }
        return nearPoints;
}
function naive_closest_point(node, point, depth = 0, best = null) {
        if (node != null) {
                var dis = distanceSquared(node.point, point);
                console.log(dis);
```

```
                if (best != null && distanceSquared(best, point) < dis) {
                        return best;
                }
                else {
                        if (node.point[node.axis] > point[node.axis]) {
                                return naive_closest_point(node.left, point, depth
                                     + 1, node.point);
                        }
                        else {
                                return naive_closest_point(node.right, point,
                                    depth + 1, node.point);
                        }
                }
        }
        else {
                return best;
        }
}
```

Tarea 5: Evalue el resultado de las dos funciones implementadas anteriormente con este conjunto de datos.

```
vardata=[
  [40,70],
  [70 ,130],
  [90,40],
  [110, 100],
  [140 ,110],
  [160, 100]
  ];
var point=[140 ,90];//query
```

Tarea 6: Evalue el resultado de las dos funciones implementadas anteriormente con este conjunto de datos.

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

Tarea 7: Ahora implemente la funcion closest point, siguiendo las recomendaciones dadas por el docente.

```
function closest_point(node, point, depth = 0) {

        if (node === null)
                return null;
        var axis = depth % k;
        var next_branch = null;
        var opposite_branch = null;
        if (point[axis] < node.point[axis]) {
                next_branch = node.left;
```

```
                opposite_branch = node.right;
        } else {
                next_branch = node.right;
                opposite_branch = node.left;
        }
        var best = closer_point(point, node, closest_point(next_branch, point,
            depth + 1));
        if (distanceSquared(point, best.point) > Math.abs(point[axis] - node.point
            [axis])) {
                best = closer_point(point, best, closest_point(opposite_branch,
                    point, depth + 1));
        }

        return best;
}

function closer_point(point, p1, p2) {
        if (p2 == null) {
                return p1;
        }
        var distance = distanceSquared(p1.point, point);
        if (distance < distanceSquared(p2.point, point))
                return p1;
        return p2;
}
```

Tarea 8: Averigue e implemente una funcion KNN, que retorna los k puntos mas cercanos a un punto.

```
function KNN(points, point, K) {
        var nearPoints = [];
        var found = [];
        for (var i = 0; i < points.length; i++)
        {
                var aux=distanceSquared(points[i],point);
                nearPoints.push([aux,points[i]])

                nearPoints.sort(function (a,b){
                        return a[0]-b[0];
                });
        }
        for(var i = 0; i < nearPoints.length; i++){
                found.push(nearPoints[i].slice(1,2));
        }
        console.log(found.slice(0, K))
}
```

Tarea 9: Implemente la funcion range query circle del KD-Tree.

```
function range_query_circle(node, center, radio, queue, depth = 0) {
        if (node == null) {
                return null;
        }
        p += 1;
        console.log(p);
        var axis = node.axis;
        var next_branch = null;
```

```
        var opposite_branch = null;
        if (center[axis] < node.point[axis]) {
                next_branch = node.left;
                opposite_branch = node.right;
        } else {
                next_branch = node.right;
                opposite_branch = node.left;
        }
        var best = closer_point(center, node, range_query_circle(next_branch,
            center, radio, queue, depth + 1));
        if (Math.abs(center[axis] - node.point[axis]) <= radio || distanceSquared(
            center, best.point) > Math.abs(center[axis] - node.point[axis])) {
                if (distanceSquared(center, node.point) <= radio) {
                        queue.push(node.point);
                }
                best = closer_point(center, best, range_query_circle(
                    opposite_branch, center, radio, queue, depth + 1));
        }

        return best;
}
```

Tarea 10: Implemente la funcion range query rec del KD-Tree, esta vez el range representa un rectangulo.

```
function range_query_rect(node , rect , found , depth = 0) {
        if (node === null) {
        return;
    }
    var axis = depth % k;
    if (node.point[axis] < rect[axis][0]){
        range_query_rect(node.right, rect, found, depth+1)
    }
    if (node.point[axis] > rect[axis][1]) {
        range_query_rect(node.left, rect, found, depth+1)
    }
    let x = node.point[0]
    let y = node.point[1]
    if (!(rect[0][0]>x || rect[0][1]<x || rect[1][0]>y || rect[1][1]<y)) {
        found.push(node.point)
    }
    range_query_rect(node.left, rect, found, depth+1)
    range_query_rect(node.right, rect, found, depth+1)
}
```

4. Resultados

Tarea 2: Cree un archivo kdtree.js.

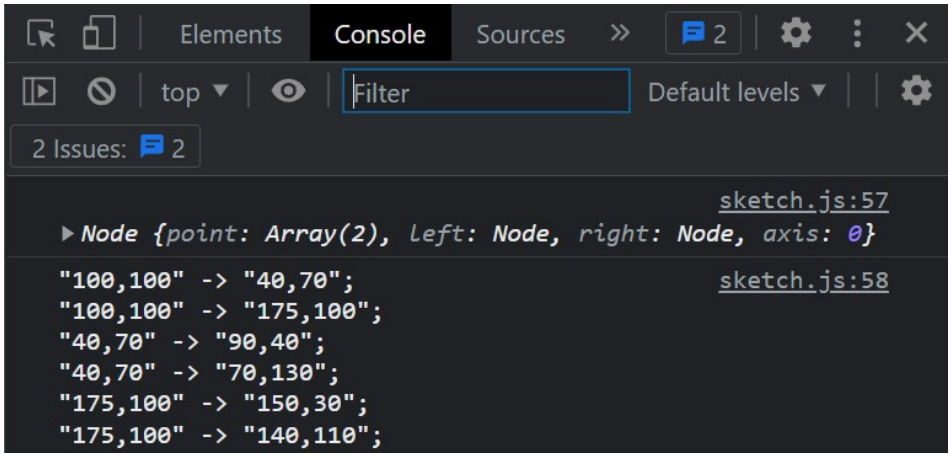

Figura 1: Resultados por consola

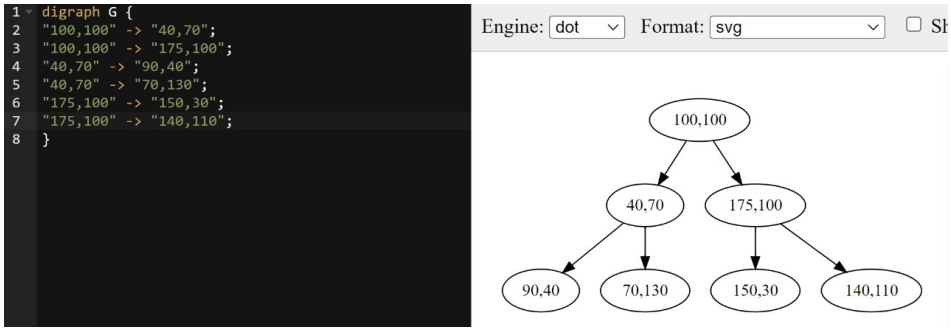

Figura 2: Vizualización de Resultados

Tarea 3: Cree un archivo sketch.js y evalue sus resultados.

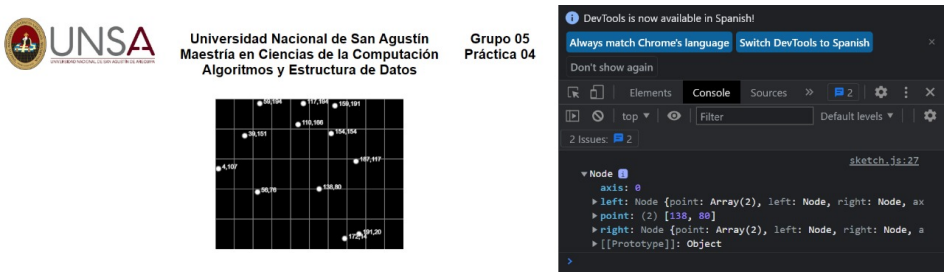

Figura 3: Vizualización de Resultados

Tarea 5: Evalue el resultado de las dos funciones implementadas anteriormente con este conjunto de datos Se ingresaron los datos sugeridos por la practica y se evaluó los resultados.

```
vardata=[
  [40,70],
  [70 ,130],
  [90,40],
```

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

Figura 4: Vizualización de Resultados

Tarea 6: Evalue el resultado de las dos funciones implementadas anteriormente con este conjunto de datos. Se ingresaron los datos sugeridos por la practica y se evaluó los resultados.

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

Figura 5: Vizualización de Resultados

Tarea 7: Ahora implemente la funcion closest point, siguiendo las recomendaciones dadas por el docente. Reconoce el punto mas cercano.

Figura 6: Vizualización de Resultados

Tarea 8: Averigue e implemente una funcion KNN, que retorna los k puntos mas cercanos a un punto. Muestra y reconoce los 4 puntos mas cercanos.

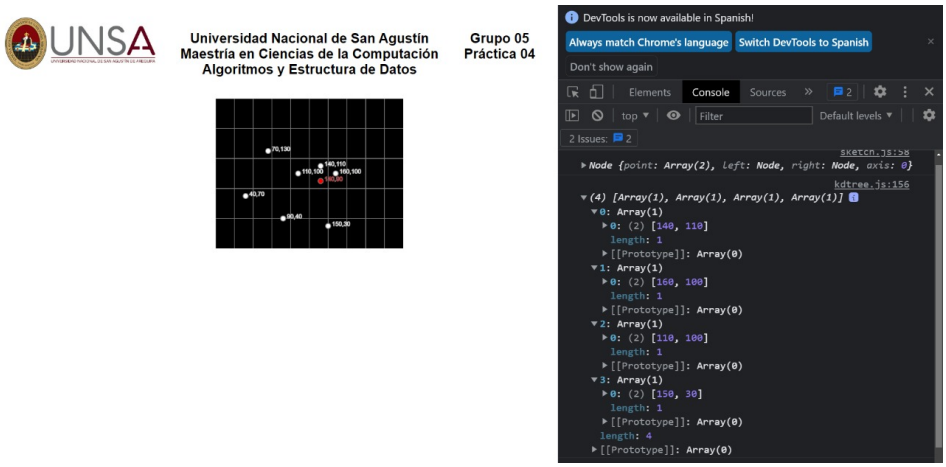

Figura 7: Vizualización de Resultados

Tarea 9: Implemente la funcion range query circle del KD-Tree. Grafica un circulo y reconoce los puntos dentro de el, en el sketch se define el area de consulta.

```
var found = [];
        var pon = [140, 100];
        var radio = 90;

    range_query_circle(root,pon,radio,found);
    console.log(found);
        fill(0, 0, 255, 40);
        circle(pon[0], height - pon[1], radio * 2);
    for ( let i = 0; i < found.length; i ++) {
        var x = found[i][0];
        var y = found[i][1];
        fill (0 , 0 , 255);
        circle (x, height - y, 7);
    }
```

Figura 8: Vizualización de Resultados

Tarea 10: Implemente la funcion range query rec del KD-Tree, esta vez el range representa un rectangulo.

Grafica un rectangulo y reconoce los puntos dentro de el, en el sketch se define el area de consulta (Rectangulo o en nuestro caso un cuadrado).

```
stroke(0,0,255);
    rectMode(CENTER);
    fill(0, 0, 255, 40);
    var rectWidth = 100
    var rectHeight = 100

    let center = [140,100];
    rect(center[0], center[1], rectWidth, rectHeight);
    var xMin = center[0] - rectWidth/2;
    var yMin = height - (center[1] + rectHeight/2);
    var xMax = center[0] + rectWidth/2;
    var yMax = height - (center[1] - rectHeight/2);

    let found = []
    var rectangle = [[xMin, xMax], [yMin, yMax]]
    range_query_rect(root, rectangle, found)
    console.log(found)
    for ( let i = 0; i < found.length; i ++) {
        var x = found[i][0];
        var y = found[i][1];
        fill (0 , 0 , 255);
        circle (x, height - y, 7);
    }
```

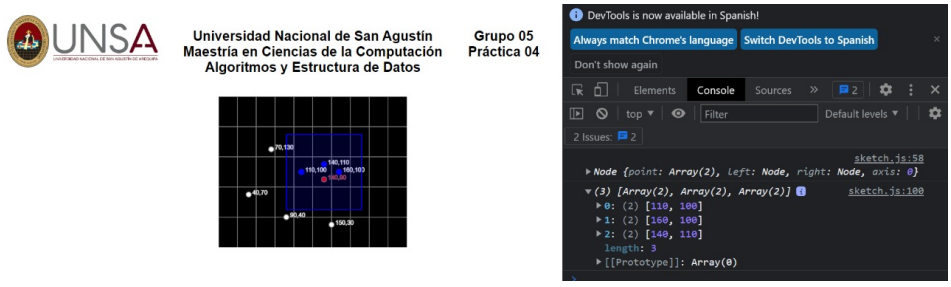

Figura 9: Vizualización de Resultados

5. Conclusiones

- 1 Comprobamos que el algoritmo naive closes point, presenta errores al no considerar algunos puntos cercanos.
- 2 Analizamos estructuras como puntos, circulos y rectángulos gracias a la librería p5.js
- 3 La recursividad nos facilitó el trabajo para implementar la mayoría de estructuras de datos solicitadas.