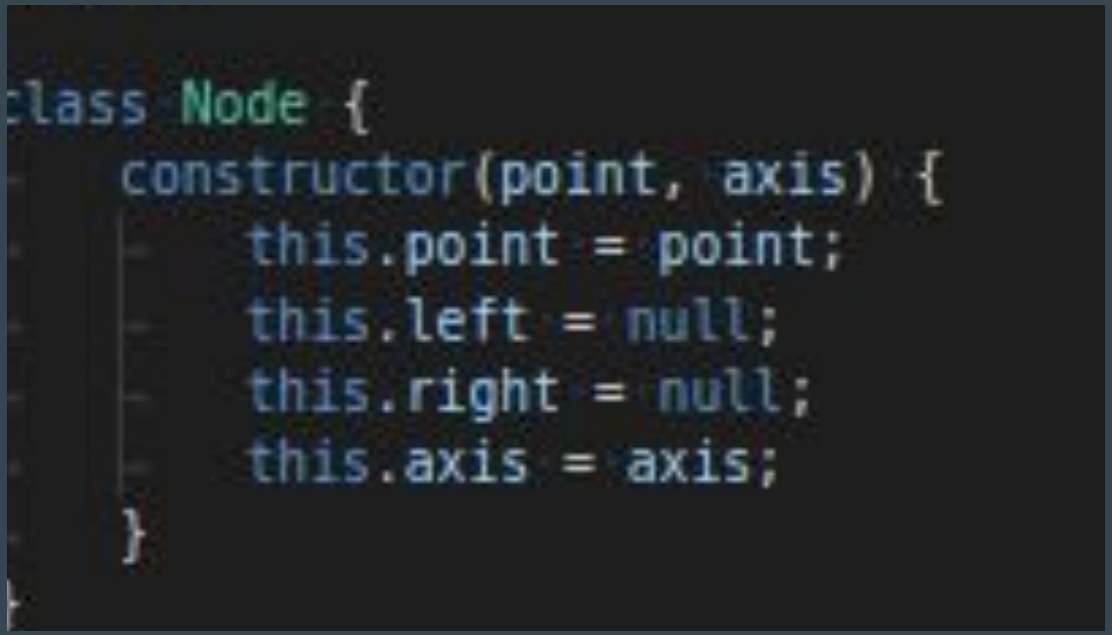
Universidad Nacional de San Agustín de Arequipa
Estructuras de Datos Avanzadas

K-d tree

...

Integrantes:
Portocarrero Espirilla, Diego Armando
Coayla Zuñiga, Gonzalo Eduardo
Jara Huilca, Arturo Jesús
Pucho Zevallos, Paul Kelvin

Clase nodo del kd tree



Construcción del kd tree

```
function build_kdtree(points, depth = 0,father = null) {
    if (!points.length) return null;

    n = points.length;
    m = points[0].length;

    eje = depth % m;

    points.sort((a, b) => a[eje] - b[eje]);
    median = Math.ceil((points.length - 1) / 2);

    let izq = [];
    let der = [];
    for (let i = 0; i < median; i++) izq.push(points[i]);
    for (let i = median + 1; i < n; i++) der.push(points[i]);

    let node = new Node(points[median], eje);

    /******************creacion de sectores***************/
    var width = 250;
    var height = 200;
    var c = color(255, 204, 0);
    stroke(c);
    if(eje == 1){
        var y = node.point[eje];
        if(node.point[father.axis] < father.point[father.axis]){
            line(0, 200-y, father.point[father.axis], 200-y);
        }else{
            line(father.point[father.axis], 200-y, width, 200-y);
        }
    }else if( eje == 0){
        var x = node.point[eje];
        if(!father){
            line(x, 0, x, height);
        }else{
            if(node.point[father.axis] < father.point[father.axis]){
                line(x, 200 - father.point[father.axis], x, height);
            }else{
                line(x, 0, x,200 - father.point[father.axis]);
            }
        }

    }
    father = node;
    /*************************************************/

    node.left = build_kdtree(izq, depth + 1,father);
    node.right = build_kdtree(der, depth + 1,father);

    return node;
}
```

closest_point_brute_force()

```
function closest_point_brute_force(points, point) {
    if (points.length < 1) return null;
    if (points.length == 1) return points[0];

    var min = distanceSquared(point, points[0]);
    var minPoint = points[0];

    for (let i = 1; i < points.length; i++) {
        var distance = distanceSquared(point, points[i]);
        if (distance < min) {
            min = distance;
            minPoint = points[i];
        }
    }

    return minPoint;
}
```

naive_closest_point()

```
function naive_closest_point(node, point, depth = 0, best = null) {
    if (!node) return best;

    if (!depth) {
        best = node.point;
    } else {
        if (distanceSquared(node.point, point) < distanceSquared(best, point)) {
            best = node.point;
        }
    }

    var axis = depth % node.point.length;

    if (point[axis] < node.point[axis]) {
        return naive_closest_point(node.left, point, depth + 1, best);
    } else {
        return naive_closest_point(node.right, point, depth + 1, best);
    }
}
```

closest_point()

```
function closest_point(node, point, depth = 0, best = null) {
    if (!node) return best;

    if (!depth) {
        best = node.point;
    } else {
        if (distanceSquared(node.point, point) < distanceSquared(best, point)) {
            best = node.point;
        }
    }
    var axis = depth % node.point.length;

    if (point[axis] < node.point[axis]) {
        best = closest_point(node.left, point, depth + 1, best);
        if (
            Math.abs(point[axis] - node.point[axis]) <
            distanceSquared(point, best)
        )
            best = closest_point(node.right, point, depth + 1, best);
    } else {
        best = closest_point(node.right, point, depth + 1, best);
        if (
            Math.abs(point[axis] - node.point[axis]) <
            distanceSquared(point, best)
        )
            best = closest_point(node.left, point, depth + 1, best);
    }
    return best;
}

function k_nearest_neighbor(node, point, arr, depth = 0, best = null) {
```

KNN(k_nearest_neighbor)

```
function k_nearest_neighbor(node, point,arr, depth = 0, best = null) {
    if (!node) return best;

    if (!depth) {
        best = node.point;
    } else {
        if (distanceSquared(node.point, point) < distanceSquared(best, point)) {
            best = node.point;
        }
    }
    var axis = depth % node.point.length;
    arr.push(node.point);
    if (point[axis] < node.point[axis]) {
        best = k_nearest_neighbor(node.left, point,arr, depth + 1, best);
        if (
            Math.abs(point[axis] - node.point[axis]) <
            distanceSquared(point, best)
        )
            best = k_nearest_neighbor(node.right, point,arr, depth + 1, best);
    } else {
        best = k_nearest_neighbor(node.right, point,arr, depth + 1, best);
        if (
            Math.abs(point[axis] - node.point[axis]) <
            distanceSquared(point, best)
        )
            best = k_nearest_neighbor(node.left, point,arr, depth + 1, best);
    }
    return best;
}
```

generate_dot()

```
function generate_dot(node) {
    string = "digraph G {\n";
    string = string + recursive_generate_dot(node);
    string = string + "}\n";
    return string;
}
```

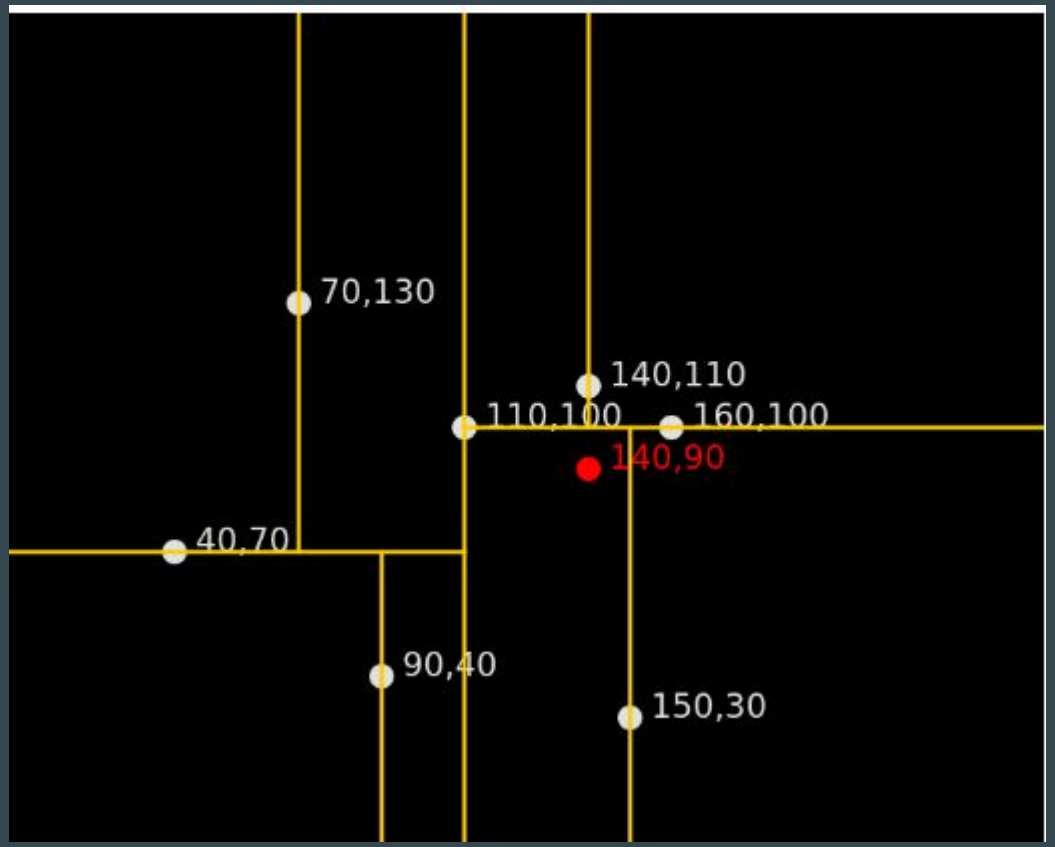
```
function recursive_generate_dot(node) {
    let txt = "";
    if (node) {
        if (node.left) {
            txt = txt + '\t"';
            txt = txt + node.point;
            txt = txt + '" -> "';
            txt = txt + node.left.point;
            txt = txt + '";\n';
            txt = txt + recursive_generate_dot(node.left);
        }
        if (node.right) {
            txt = txt + '\t"';
            txt = txt + node.point;
            txt = txt + '" -> "';
            txt = txt + node.right.point;
            txt = txt + '";\n';
            txt = txt + recursive_generate_dot(node.right);
        }
    }
    return txt;
}

function generate_dot(node) {
    string = "digraph G {\n";
    string = string + recursive_generate_dot(node);
    string = string + "}\n";
    return string;
}
```

Resultados

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

```
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";
}
```

Resultados




```
Queried point: 140,90
Brute force: 140,110
Naive closest point: 160,100
Closest point: 140,110
Más cercanos:
▸ Array [ 140, 110 ]
▸ Array [ 160, 100 ]
▸ Array [ 110, 100 ]
```

Resultados