UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERIA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESINAL DE INGENIERIA DE SISTEMAS

KD - TREES

CURSO:

ESTRUCTURAS DE DATOS AVANZADAS

DOCENTE:

ING. COLLANQUI MARTINEZ FREDY

PRESENTADO:

POR ELI RONNIE QUISPE TICONA

SEMESTRE:

Sexto

GRUPO:

C

2024

PUNO - PERÚ

Índice

1. Introducción

El uso de teléfonos móviles para gestionar agendas con fotos y comandos de voz ahora es posible gracias a los importantes avances en las tecnologías de la información y las comunicaciones en la última década y es la nueva forma de escribir direcciones es convertirlas en enlaces a sitios web que muestran dónde están las calles, cómo llegar y qué tipo de transporte puede utilizar.

Para manejar toda esta información utilizamos estructuras de datos como árboles multidimensionales, que son muy importantes en las bases de datos para responder diferentes tipos de consultas, desde coincidencias exactas hasta búsquedas de elementos similares o cercanos que cumplan múltiples condiciones.

Los KD-Trees es fantástico para encontrar puntos cercanos, realizar búsquedas de rango o encontrar el punto más cercano en un conjunto de datos, lo cual permite dividir las cosas en partes más pequeñas hace que sea mucho más rápido encontrarlas y obtenemos una estructura equilibrada que facilita la colocación de datos en grandes espacios.

2. Capítulo 1

3. Kd-trees

Los kd-trees son una versión avanzada de los árboles binarios de búsqueda que permiten trabajar con datos que tienen varias dimensiones, los algoritmos para actualizar estos árboles pueden ser bastante complicados y para simplificar significativamente los procesos de inserción y eliminación, se añadirá un nuevo componente llamado criterio a la clave de los kd-trees lo cual esto da lugar a un nuevo tipo de árbol binario de búsqueda multidimensional, conocido como relaxed kd-tree (de Berg, 2008).

3.1. Kd – tree Bidimensional

Los kd-trees tienen como clave un registro multidimensional, es decir, cada clave multidimensional x está formada por k valores de claves unidimensionales, siendo k la dimensión del espacio:

$$x = x_1, x_2, \dots, x_k$$

Esta clave multidimensional puede ser vista como un punto del espacio de k dimensiones y cada componente unidimensional, x_i, como la coordenada j-ésima del punto.

Un kd-tree, árbol binario de búsqueda de k dimensiones o árbol binario multidimensional para un conjunto de claves de k dimensiones es un árbol binario de búsqueda que cumple lo siguiente:

- Cada nodo tiene como clave multidimensional un vector de tamaño k que contiene los valores de las k claves unidimensionales y tiene asociado un discriminante con valor entero entre 1 y k.
- Para cada nodo con clave multidimensional x y discriminante j se cumple que cualquier nodo del subárbol izquierdo con clave multidimensional y cumple que

su j-ésima componente es menor que la j-ésima componente de x, es decir ($y_j < x_j$). Para el subárbol derecho se cumple que la j-ésima componente de la clave multidimensional z de cualquier nodo de este subárbol es mayor que la j-ésima componente de la clave x, es decir ($z_j > x_j$).

- La raíz del kd-tree tiene profundidad 0 y discriminante 1. Cualquier nodo a una profundidad p tiene discriminante de valor: (p mod k) + 1.

La Figura 1 muestra un kd-tree en un espacio de dos dimensiones, donde k = 2, las claves bidimensionales x que representan un punto en el plano, están formadas por dos claves unidimensionales (x_1, x_2). Cada una de estas claves tiene asignado un valor de discriminante, que se calcula a partir de la profundidad del nodo.

El discriminante de la raíz es 1, por tanto, en el subárbol izquierdo estarán todas las claves bidimensionales cuya primera componente cumpla x1 < 0.5; de manera análoga en el subárbol derecho estarán las claves que cumplan x1 > 0.5.

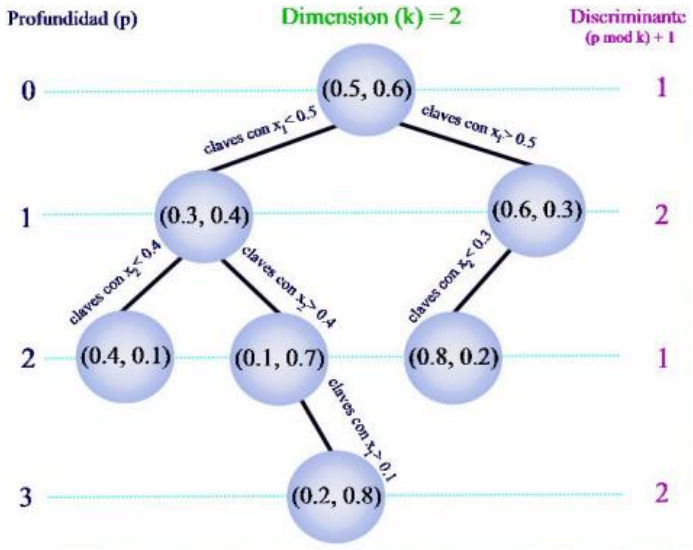

Figura 1: Un árbol binario de búsqueda de 2 dimensiones ó de 2d-tree.

El discriminante de la raíz del subárbol izquierdo, (0.3, 0.4), es 2, luego será la segunda componente de la clave la que discrimine a cuál de sus hijos, izquierdo o derecho,

corresponden las claves de sus hijos, así, en su hijo izquierdo estarán todos los nodos con valores de clave cuya segunda componente cumpla < 0.4 y además x_1 < 0.5 (discriminado por el padre de (0.3, 0.4)); a su vez en el hijo derecho estarán los nodos con claves tales que x_2 > 0.4 y además x_1 < 0.5 y así sucesivamente.

Desde el punto de vista geométrico, cada nodo del 2d-tree hace una partición del plano en dos "subplanos" según la recta perpendicular al eje determinada por la componente que indica el discriminante del nodo. En la Figura 2 todos los puntos en el "subplano" de la izquierda corresponden al subárbol izquierdo del nodo raíz, que discrimina según el eje X, del árbol de la Figura 1 y los que quedan a la derecha son los de su subárbol derecho. (Berlin Heidelberg, 2013)

De la misma manera, para un 3d-tree tendremos un espacio cúbico de k = 3, particionado por planos perpendiculares a la dimensión correspondiente a la discriminante. En el caso de una sola dimensión, k = 1, estaremos sobre una recta, en este caso, el 1d-tree no es más que un árbol binario de búsqueda. En general, para valores de k > 3 tendremos particiones del hiperespacio (forma de espacio que tiene cuatro o más dimensiones) de dimensión k según el hiperplano determinado por el discriminante de cada clave multidimensional.

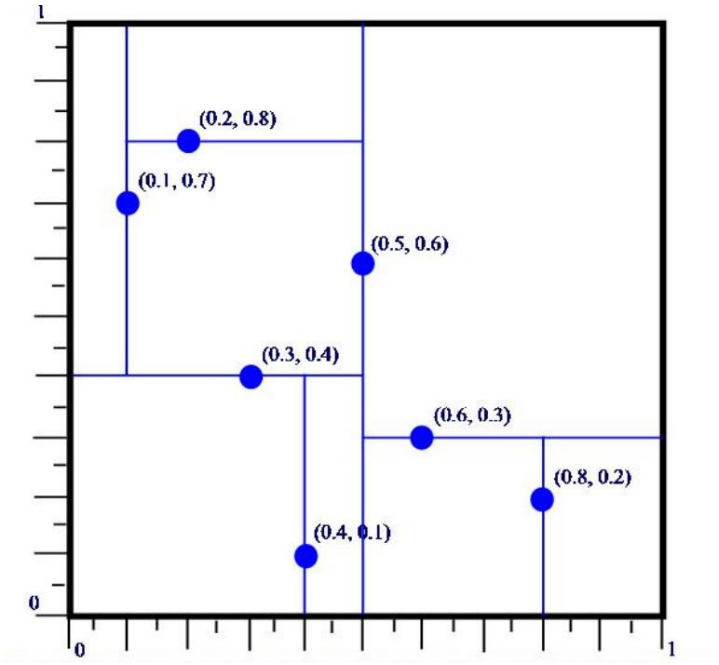

Figura 2: Particiones del plano debidas a las claves del kd-tree de la Figura 1.

Los kd-trees se utilizan frecuentemente en bases de datos para manejar consultas que involucran múltiples campos, por ejemplo, una consulta típica podría ser encontrar todos los registros de personas que tengan entre 24 y 49 años y cuyos ingresos anuales estén entre 40.000 y 70.000 euros. Esto permite realizar búsquedas eficientes en grandes conjuntos de datos, utilizando las múltiples dimensiones de la información almacenada.

Otra aplicación importante de los kd-trees es en la representación de conjuntos de puntos en el espacio multidimensional, una consulta muy común en este contexto es la búsqueda del punto o puntos más cercanos a un punto dado, conocida como búsqueda del vecino más cercano. Esta operación es crucial en diversos campos como la computación gráfica, la inteligencia artificial y el análisis de datos, donde es importante identificar rápidamente los elementos más cercanos en términos de distancia espacial o de similitud.

Además, los kd-trees son especialmente útiles en el procesamiento de grandes volúmenes de datos en tiempo real, ya que permiten realizar consultas rápidas y

eficientes sin necesidad de recorrer exhaustivamente toda la base de datos y esto los hace ideales para aplicaciones en sistemas de recomendación, motores de búsqueda, y otras áreas donde la rapidez y precisión en las búsquedas son esenciales.

3.2. Kd - tree Tridimencional

El árbol kd tridimensional es una estructura de datos versátil y eficiente para la organización y búsqueda de puntos en un espacio 3D, su capacidad para realizar búsquedas rápidas y su flexibilidad en aplicaciones multidimensionales lo hacen ideal para una amplia gama de aplicaciones prácticas, desde gráficos por computadora hasta análisis de datos y aprendizaje automático (Gonzalez, Albusac, & Perez, 2019).

Se divide en:

- Nodos: Cada nodo en un árbol kd representa un punto en el espacio k-dimensional (en este caso, 3D). Además de almacenar el punto, cada nodo tiene punteros a sus hijos izquierdo y derecho.
- División del Espacio: El árbol kd divide recursivamente el espacio en regiones más pequeñas utilizando hiperplanos perpendiculares a uno de los ejes de coordenadas (x, y, z).

Construcción del Árbol

- Eje de División: El eje de división alterna en cada nivel del árbol, en un árbol 3D, la secuencia de ejes es x, y, z, x, y, z, y así sucesivamente.
- Punto Mediano: Para equilibrar el árbol, se elige el punto mediano de los puntos ordenados según el eje de división actual, este punto se convierte en el nodo del árbol y los puntos restantes se dividen en subárboles izquierdo y derecho.

Aplicaciones Avanzadas del Árbol kd

Búsqueda de Vecinos Más Cercanos (Nearest Neighbor Search)

Utiliza el árbol kd para encontrar rápidamente el punto más cercano a un punto dado en el espacio 3D.

- Algoritmo: Inicia la búsqueda desde la raíz, comparando el punto de consulta con el nodo actual y recursivamente visita el subárbol que contiene al punto de consulta. Se usa una búsqueda de retroceso para verificar si es necesario visitar el otro subárbol.

Búsqueda de Rango (Range Search)

Encuentra todos los puntos dentro de un rango especificado.

- Algoritmo: Realiza una búsqueda recursiva, comparando los límites del rango con los nodos del árbol y visitando los subárboles correspondientes.

Detección de Colisiones

Utiliza el árbol kd para detectar colisiones entre objetos en simulaciones físicas y gráficos por computadora.

- Algoritmo: Almacena las posiciones de los objetos en el árbol kd y utiliza búsquedas de rango para detectar posibles colisiones.

Agrupación de Datos (Clustering)

Facilita la agrupación de puntos en clústeres basados en proximidad espacial.

- Algoritmo: Utiliza el árbol kd para realizar búsquedas eficientes de vecinos más cercanos, agrupando puntos cercanos en clústeres.

En el siguiente imagen observamos la forma que tiene un árbol en tres dimensiones en cual se puede observar que se tiene particiones en tres ejes que viene a ser (x,y,z) a diferencia al 2d tree:

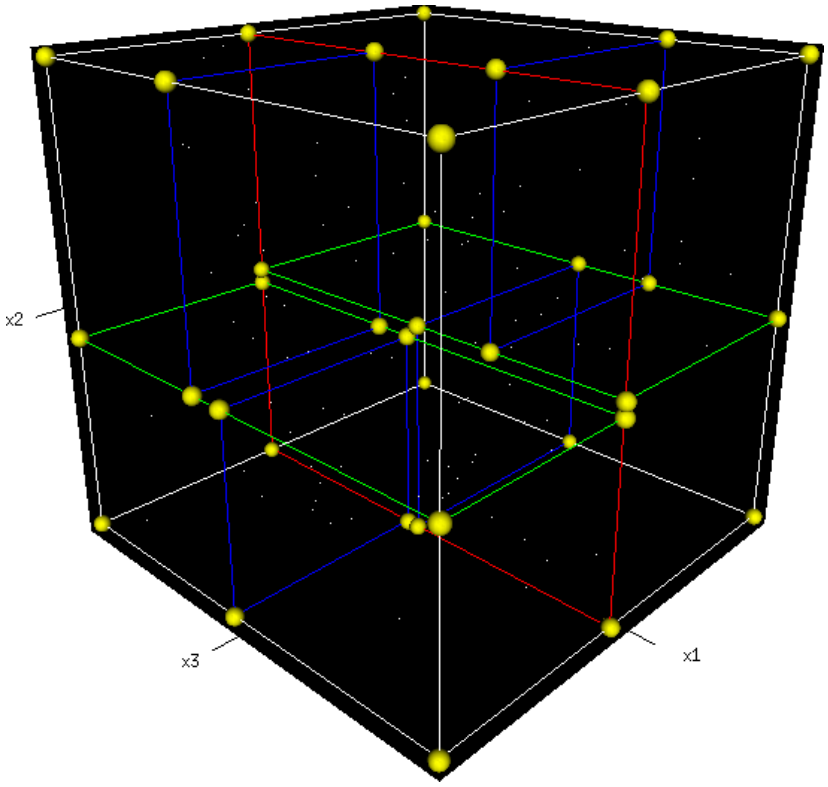

Figura 2: Ejemplo de un árbol kd tridimensional.

3.3. Relaxed kd-trees

Los relaxed kd-trees (árboles kd relajados) son una variación avanzada de los kd-trees tradicionales, diseñados para manejar de manera eficiente elementos con claves multidimensionales, esta estructura de datos introduce un componente adicional llamado discriminante, que simplifica significativamente los algoritmos de inserción y eliminación. Gracias a esta característica, los relaxed kd-trees pueden ejecutar cualquier secuencia de operaciones de actualización manteniendo la eficiencia y aleatoriedad del árbol (H. Möhring & Raman, 2002).

Los relaxed kd-trees, árboles binarios de búsqueda de k dimensiones relajados en cada nodo almacena explícitamente su discriminante y la secuencia de discriminantes desde la raíz del árbol a cualquiera de sus hojas es arbitraria.

Más formalmente, un relaxed kd-tree para un conjunto de claves k-dimensionales es un árbol binario que cumple lo siguiente:

- Cada nodo contiene una clave de dimensión k y tiene asociado un discriminante j ∈ {1, 2, . . . , k}.
- Para cada nodo con clave x y discriminante j cualquier nodo de su subárbol izquierdo con clave y cumple $y_j < x_j$ y cualquier nodo de su subárbol derecho con clave z cumple $z_j > x_j$.

La Figura 3 es un relaxed kd-tree de dos dimensiones en el que cada nodo contiene su correspondiente par [clave, discriminante]; en la Figura 4 se muestra la partición del espacio de genera. Recordar que un relaxed kd-tree de una dimensión, es un árbol binario de búsqueda y que si el mismo conjunto de claves es introducido en orden diferente resulta un relaxed kd-tree diferente, también resulta un árbol diferente si para el mismo orden de entrada de claves se eligen diferentes discriminantes.

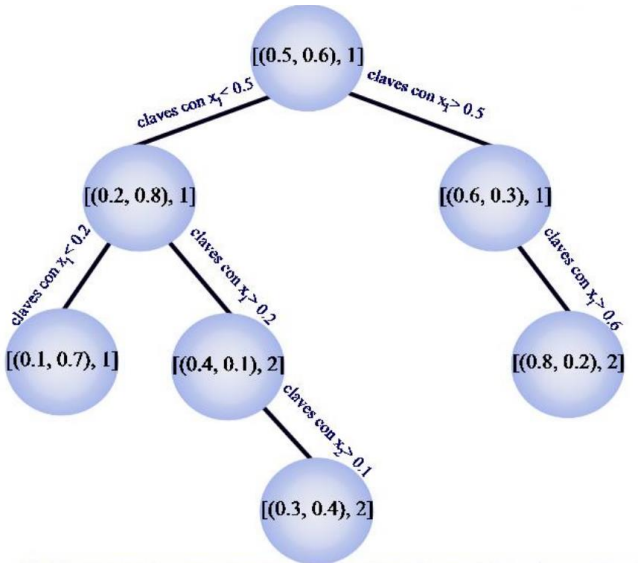

Figura 4: Un relaxed 2d-tree.

Lo cual genera el siguiente resultado de las reparticiones de la Figura 3:

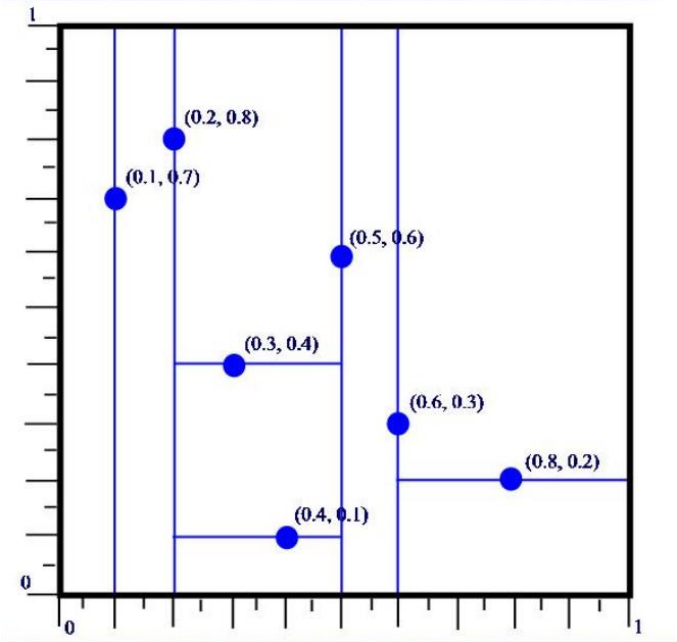

Figura 5: Plano con las particiones correspondientes al relaxed kd-tree de la Figura 3.

3.4. Kd-trees aleatorios

Los kd-trees aleatorios son una variación de los tradicionales kd-trees que incorporan aleatorización en su construcción para mejorar su rendimiento y evitar problemas de degeneración, a diferencia de los kd-trees convencionales, donde la estructura del árbol

puede volverse desequilibrada con el tiempo debido a inserciones y eliminaciones sucesivas, aparte de eso introducen una selección aleatoria de los ejes y puntos de partición para mantener el árbol balanceado de manera más natural (Jain, 2009).

En un kd-tree aleatorio, el eje de partición en cada nivel del árbol se elige de forma aleatoria en lugar de seguir un ciclo fijo únicamente en la varianza de los datos. Además, el punto de partición, que determina cómo se dividen los nodos, también se selecciona aleatoriamente de entre los puntos disponibles en el conjunto de datos actual y este enfoque reduce la posibilidad de que se formen ramas desbalanceadas, lo que puede ocurrir en kd-trees tradicionales cuando los datos no están distribuidos uniformemente (J. Goldman & A. Goldman, , 2007).

Un árbol binario aleatorio de k dimensiones, conocido como kd-tree aleatorizado, presenta las siguientes características:

- Permite realizar cualquier secuencia de operaciones de actualización, inserciones y eliminaciones, manteniendo la aleatoriedad del árbol.
- No requiere ningún tipo de preprocesamiento de los datos.
- Es eficiente en operaciones de búsqueda por clave y consultas asociativas.

El uso de kd-trees aleatorios tiene varias ventajas:

Eficiencia de Consulta: La estructura más equilibrada resultante de la aleatorización mejora la eficiencia de las consultas, ya que reduce la profundidad promedio del árbol y, por lo tanto, el número de nodos que deben ser visitados.

Robustez: La aleatorización hace que el árbol sea más robusto frente a patrones específicos en los datos, evitando casos extremos donde el rendimiento del árbol podría degradarse significativamente.

Simetría y Equilibrio: La probabilidad de obtener un árbol simétrico y equilibrado es mayor, lo que favorece la distribución uniforme de las cargas de búsqueda y actualización.

3.5. Construcción de un kd-tree

Los kd-trees son ampliamente utilizados en problemas que involucran consultas espaciales, tales como la búsqueda de vecinos más cercanos, la búsqueda por rangos y la organización de bases de datos multidimensionales (Sabry, 2024).

3.5.1. Proceso de Construcción del kd-tree

La construcción de un kd-tree implica insertar puntos de manera que el árbol se mantenga balanceado y eficiente para las consultas espaciales. A continuación, se describen los pasos teóricos para la construcción de un kd-tree bidimensional (2d-tree):

3.5.2. Inicialización

Comenzamos con un árbol vacío y esta vez el árbol se inicializa sin nodos.

3.5.3. Inserción de Puntos

Punto de Partida: El primer punto insertado se convierte en la raíz del árbol.

Alternancia de Ejes: A medida que se insertan más puntos, el eje de división alterna entre las dimensiones. En un 2d-tree, se alterna entre el eje x y el eje y.

Comparación y Posicionamiento: Para cada nuevo punto, se compara la coordenada del punto correspondiente al eje actual con la coordenada del nodo actual.

- Si el nuevo punto es menor, se desciende al subárbol izquierdo.
- Si es mayor o igual, se desciende al subárbol derecho.

Nodo Hoja: Cuando se alcanza un nodo hoja (sin hijos), el nuevo punto se inserta en esa posición.

3.5.4. Partición del Espacio

- Cada nodo en el kd-tree divide el espacio en dos mitades a lo largo de la dimensión del eje actual.
- Los nodos de niveles pares dividen a lo largo del eje x, mientras que los nodos de niveles impares dividen a lo largo del eje y.
- Esta partición del espacio asegura que cada punto se coloque en una región específica del espacio, facilitando las consultas espaciales.

3.6. Propiedades del kd-tree

3.6.1. Balance y Eficiencia

- Los kd-trees son eficientes para consultas espaciales debido a su estructura balanceada. La alternancia de ejes y la partición del espacio permiten reducir el número de comparaciones necesarias para encontrar un punto.
- En el peor de los casos, la profundidad del árbol es O(log n) donde n es el número de puntos, lo que permite búsquedas eficientes.

1.1.1. Operaciones de Búsqueda

- Búsqueda de Vecino Más Cercano: Se utiliza para encontrar el punto más cercano a un punto dado en el espacio. El algoritmo recorre el árbol, comparando distancias y eliminando subárboles que no pueden contener un punto más cercano que el mejor encontrado hasta el momento.
- Búsqueda por Rangos: Permite encontrar todos los puntos dentro de un rango especificado. Se realiza un recorrido del árbol, comprobando qué subárboles intersecan con el rango de búsqueda.

1.1.2. Inserción y Eliminación

- La inserción de nuevos puntos se realiza de manera recursiva, siguiendo las reglas de comparación y posicionamiento descritas anteriormente.
- La eliminación de puntos es más compleja, ya que puede requerir la reestructuración del árbol para mantener su balance y las particiones del espacio correctamente organizadas.

4. Implementación

Para demostrar lo teórico se realizo una demostración en JavaScript para ver y generar como es que funciona la búsqueda de k dimensiones, esta vez se implemento solamente un algoritmo de kd-tree bidimensional para observar el funcionamiento.

4.1. Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>K-D Tree Canvas</title>
    <style>
        #canvas {
            border: 1px solid black;
        }
    </style>
</head>
<body>
    <canvas id="canvas" width="1400" height="550"></canvas>
    <div>
        X: <input type="text" id="mouseX" readonly>
        Y: <input type="text" id="mouseY" readonly>
    </div>
    <script>
        class Punto {
            constructor(x = 0, y = 0) {
                this.x = x;
                this.y = y;
            }
        }

        class Nodo {
            constructor(P, eje = 0, Izq = null, Der = null) {
                this.P = P;
                this.eje = eje;
                this.Izq = Izq;
```

```
            this.Der = Der;
        }
    }

    class k2dTree {
        constructor() {
            this.Raiz = null;
        }

        Insertar(PP) {
            this.Raiz = this.Ins(PP, 0, this.Raiz);
        }

        Ins(PP, e, R) {
            if (R == null) {
                return new Nodo(PP, e);
            } else {
                if (e == 0) {
                    if (PP.x < R.P.x) {
                        R.Izq = this.Ins(PP, 1, R.Izq);
                    } else {
                        R.Der = this.Ins(PP, 1, R.Der);
                    }
                } else {
                    if (PP.y < R.P.y) {
                        R.Izq = this.Ins(PP, 0, R.Izq);
                    } else {
                        R.Der = this.Ins(PP, 0, R.Der);
                    }
                }
                return R;
            }
        }

        MostrarR(ctx, x, y, a) {
            this.MosR(this.Raiz, ctx, x, y, a);
        }

        MosR(R, ctx, x, y, a) {
            if (R != null) {
                const nodeRadius = 15;
                ctx.beginPath();
                ctx.arc(x, y, nodeRadius, 0, 2 * Math.PI);
                ctx.fillStyle = "lightblue";
                ctx.fill();
                ctx.stroke();
                ctx.fillStyle = "black";
                ctx.fillText(`(${R.P.x}, ${R.P.y})`, x -
nodeRadius, y - nodeRadius - 5);

                if (R.Izq) {
                    ctx.beginPath();
                    ctx.moveTo(x, y + nodeRadius);
                    ctx.lineTo(x - a, y + 50 - nodeRadius);
                    ctx.stroke();
                    this.MosR(R.Izq, ctx, x - a, y + 50, a / 2);
                }

                if (R.Der) {
                    ctx.beginPath();
                    ctx.moveTo(x, y + nodeRadius);
```

```
                    ctx.lineTo(x + a, y + 50 - nodeRadius);
                    ctx.stroke();
                    this.MosR(R.Der, ctx, x + a, y + 50, a / 2);
                }
            }
        }

        Mostrar(ctx, xi, yi, xf, yf) {
            this.Mos(this.Raiz, ctx, xi, yi, xf, yf);
        }

        Mos(R, ctx, xi, yi, xf, yf) {
            if (R != null) {
                let xx = R.P.x;
                let yy = R.P.y;
                if (R.eje == 1) {
                    ctx.beginPath();
                    ctx.moveTo(xi, yy);
                    ctx.lineTo(xf, yy);
                    ctx.stroke();
                    this.Mos(R.Izq, ctx, xi, yi, xf, yy);
                    this.Mos(R.Der, ctx, xi, yy, xf, yf);
                } else {
                    ctx.beginPath();
                    ctx.moveTo(xx, yi);
                    ctx.lineTo(xx, yf);
                    ctx.stroke();
                    this.Mos(R.Izq, ctx, xi, yi, xx, yf);
                    this.Mos(R.Der, ctx, xx, yi, xf, yf);
                }
                ctx.beginPath();
                ctx.arc(xx, yy, 4, 0, 2 * Math.PI);
                ctx.fill();
            }
        }
    }

    const canvas = document.getElementById('canvas');
    const ctx = canvas.getContext('2d');
    const tree = new k2dTree();

    const pxi = 10, pyi = 50, pxf = 410, pyf = 450; // Cambié las
dimensiones del cuadro aquí

    canvas.addEventListener('mousemove', (event) => {
        const rect = canvas.getBoundingClientRect();
        const xPos = event.clientX - rect.left;
        const yPos = event.clientY - rect.top;
        document.getElementById('mouseX').value = xPos;
        document.getElementById('mouseY').value = yPos;
    });

    canvas.addEventListener('click', (event) => {
        const rect = canvas.getBoundingClientRect();
        const xPos = event.clientX - rect.left;
        const yPos = event.clientY - rect.top;
        if (xPos < pxf && xPos > pxi && yPos < pyf && yPos > pyi)
{
            tree.Insertar(new Punto(xPos, yPos));
            draw();
        }
```

```
        });

        function draw() {
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            ctx.strokeRect(pxi, pyi, pxf - pxi, pyf - pyi);
            tree.Mostrar(ctx, pxi, pyi, pxf, pyf);
            tree.MostrarR(ctx, 950, 50, 200); // Ajusté la posición
del árbol
        }

        draw();
    </script>
</body>
</html>
```

Lo cual nos genera un cuadro de dialogo donde dentro del cuadro podemos colocar líneas para generar o manipular un kd-tree que por esta vez es bidimensional, así como también a su costado se muestra los puntos que se generan. Solamente como un ejemplo

Lo que resulta es:

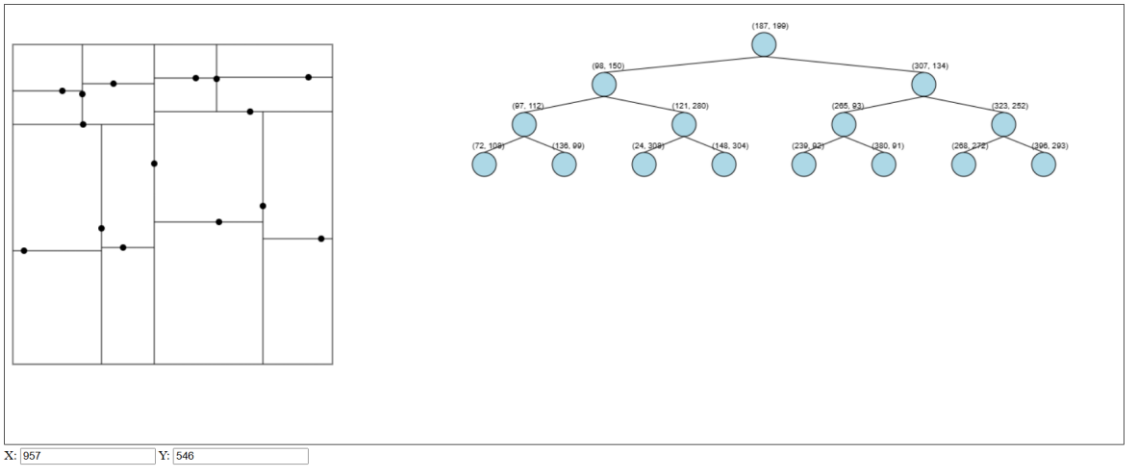

Figura 4: Resultado de kd-tree bidimensional en JavaScript.

5. Conclusión

Los kd-trees son una poderosa estructura de datos utilizada en el manejo eficiente de puntos en espacios k-dimensionales, donde comenzamos comprendiendo su construcción teórica, destacando la alternancia de ejes y la partición del espacio como elementos fundamentales en su implementación. A través de algoritmos de inserción y visualización, pudimos entender cómo los kd-trees organizan los puntos de manera equilibrada y permiten operaciones de búsqueda eficientes, como la búsqueda del vecino más cercano y las consultas por rango.

Además, discutimos la aplicación práctica de los kd-trees en diversos campos, desde sistemas de información geográfica hasta bases de datos multidimensionales, aprovechando su capacidad para manejar grandes volúmenes de datos y realizar consultas espaciales de manera rápida y precisa. En este contexto, también exploramos las variantes avanzadas de los kd-trees, como los kd-trees relajados y aleatorios, que abordan desafíos específicos relacionados con la actualización y mantenimiento de la estructura del árbol, así como la garantía de su eficiencia a lo largo del tiempo.

Podemos decir que los kd-trees representan una herramienta invaluable en el análisis y procesamiento de datos multidimensionales, ofreciendo una combinación única de eficiencia, versatilidad y capacidad de respuesta que los convierte en una elección popular en una amplia gama de aplicaciones y escenarios de ingeniería y ciencias de la computación.

6. Bibliografía

Berlin Heidelberg, S. (2013). *Advances in Soft Computing and Its Applications.* Mexico: 12th Mexican International Conference, MICAI 2013, Mexico City, Mexico, November 24-30, 2013, Proceedings, Part II.

de Berg, M. (2008). *Computational Geometry: Algorithms and Applications.* España: Springer Berlin Heidelberg.

Gonzalez, C., Albusac, J., & Perez, S. (2019). *Creación de Videojuegos en Español.* España: Cursos en Español.

H. Möhring, R., & Raman, R. (2002). *Algorithms - ESA 2002.* Italia: Springer, Annual European Symposium, Rome.

J. Goldman, K., & A. Goldman, , S. (2007). *A Practical Guide to Data Structures and Algorithms Using Java.* Italia: CRC Press.

Jain, V. (2009). *INFORMATION technology issues & challenges.* España: Excel Books: Pioneer Institute of Professional Studies, Indore.

Sabry, F. (2024). *Partición del espacio binario: Explorando la partición del espacio binario: fundamentos y aplicaciones en visión por computadora.* España: Mil Millones De Conocimientos [Spanish].