

## Práctica 04

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

PRÁCTICA	TEMA	DURACIÓN
04	Kd-tree	—

### 1. Integrantes

- Grupo N° 2
- Integrantes:
  - EDER ALONSO AMPUERO ATAMARI
  - HOWARD FERNANDO ARANZAMENDI MORALES
  - JOSE EDISON PEREZ MAMANI
  - HENRRY IVAN ARIAS MAMANI

### 2. Repositorio GitHub

URL Github: Repositorio Práctica 4 AyED

### 3. Video de Exposición en YouTube

URL YouTube: Video de Exposición en YouTube

## 4. Marco Teórico

### 4.1. KD-Tree

#### 4.1.1. Definición

Un KD-Tree (también llamado árbol K-dimensional) es un árbol de búsqueda binaria donde los datos en cada nodo son un punto K-dimensional en el espacio. En resumen, es una estructura de datos de partición de espacio (detalles a continuación) para organizar puntos en un espacio K-Dimensional.

Un nodo que no es una hoja en el árbol K-D divide el espacio en dos partes, llamadas medios espacios.

Los puntos a la izquierda de este espacio están representados por el subárbol izquierdo de ese nodo y los puntos a la derecha del espacio están representados por el subárbol derecho. Pronto estaremos explicando el concepto de cómo se divide el espacio y se forma el árbol.

En aras de la simplicidad, entendamos un árbol 2-D con un ejemplo.

La raíz tendría un plano alineado con el eje x, los hijos de la raíz tendrían ambos planos alineados con el eje y, los nietos de la raíz tendrían todos planos alineados con el eje x, y los bisnietos de la raíz tendrían todos planos alineados con el eje y, y así sucesivamente.

#### 4.1.2. Generalización

Numeremos los planos como 0, 1, 2, ... (K - 1). Del ejemplo anterior, es bastante claro que un punto (nodo) en la profundidad D tendrá un plano alineado donde A se calcula como:

$$A = D \bmod K$$

#### 4.1.3. ¿Cómo determinar si un punto estará en el subárbol izquierdo o en el subárbol derecho?

Si el nodo raíz está alineado en el plano A, el subárbol izquierdo contendrá todos los puntos cuyas coordenadas en ese plano sean más pequeñas que las del nodo raíz. De manera similar, el subárbol derecho contendrá todos los puntos cuyas coordenadas en ese plano sean mayores-iguales que las del nodo raíz.

#### 4.1.4. Creación de un árbol 2-D

Considere los siguientes puntos en un plano 2-D: (3, 6), (17, 15), (13, 15), (6, 12), (9, 1), (2, 7), (10, 19). De acuerdo a la figura 5

1. Insertar (3, 6): dado que el árbol está vacío, conviértalo en el nodo raíz.
2. Insertar (17, 15): compararlo con el punto del nodo raíz. Dado que el nodo raíz está alineado con X, el valor de la coordenada X se comparará para determinar si se encuentra en el subárbol derecho o en el subárbol izquierdo. Este punto estará alineado con Y.
3. Insertar (13, 15): el valor X de este punto es mayor que el valor X del punto en el nodo raíz. Entonces, esto estará en el subárbol derecho de (3, 6). Nuevamente compare el valor Y de este punto con el valor Y del punto (17, 15) (¿Por qué?). Como son iguales, este punto estará en el subárbol derecho de (17, 15). Este punto estará alineado con X.
4. Insertar (6, 12): el valor X de este punto es mayor que el valor X del punto en el nodo raíz. Entonces, esto estará en el subárbol derecho de (3, 6). Nuevamente compare el valor Y de este punto con el valor Y del punto (17, 15) (¿Por qué?). Como 12 < 15, este punto estará en el subárbol izquierdo de (17, 15). Este punto estará alineado con X.
5. Insertar (9, 1): De manera similar, este punto estará a la derecha de (6, 12).

6. Insertar (2, 7): De manera similar, este punto estará a la izquierda de (3, 6).
7. Inserta (10, 19): De manera similar, este punto estará a la izquierda de (13, 15).

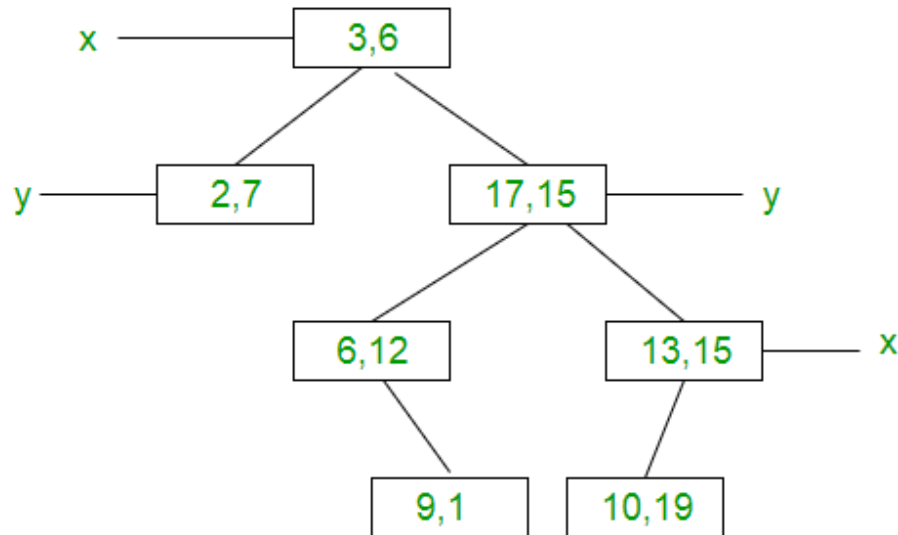


Figura 1: Nodos en KD-Tree

#### 4.1.5. ¿Cómo se divide el espacio?

Los 7 puntos se trazarán en el plano X-Y de la siguiente manera:

1. El punto (3, 6) dividirá el espacio en dos partes: Dibujar la línea  $X = 3$ .
2. El punto (2, 7) dividirá el espacio a la izquierda de la línea  $X = 3$  en dos partes horizontalmente. Dibuja la línea  $Y = 7$  a la izquierda de la línea  $X = 3$ .
3. El punto (17, 15) dividirá el espacio a la derecha de la línea  $X = 3$  en dos partes horizontalmente. Dibuja la línea  $Y = 15$  a la derecha de la línea  $X = 3$ .
4. El punto (6, 12) dividirá el espacio debajo de la línea  $Y = 15$  ya la derecha de la línea  $X = 3$  en dos partes. Dibuja la línea  $X = 6$  a la derecha de la línea  $X = 3$  y debajo de la línea  $Y = 15$ .
5. El punto (13, 15) dividirá el espacio debajo de la línea  $Y = 15$  ya la derecha de la línea  $X = 6$  en dos partes. Dibuja la línea  $X = 13$  a la derecha de la línea  $X = 6$  y debajo de la línea  $Y = 15$ .
6. El punto (9, 1) dividirá el espacio entre las líneas  $X = 3$ ,  $X = 6$  e  $Y = 15$  en dos partes. Dibuja la línea  $Y = 1$  entre las líneas  $X = 3$  y  $X = 6$ .
7. El punto (10, 19) dividirá el espacio a la derecha de la línea  $X = 3$  y arriba de la línea  $Y = 15$  en dos partes. Dibuja la línea  $Y = 19$  a la derecha de la línea  $X = 3$  y arriba de la línea  $Y = 15$ .

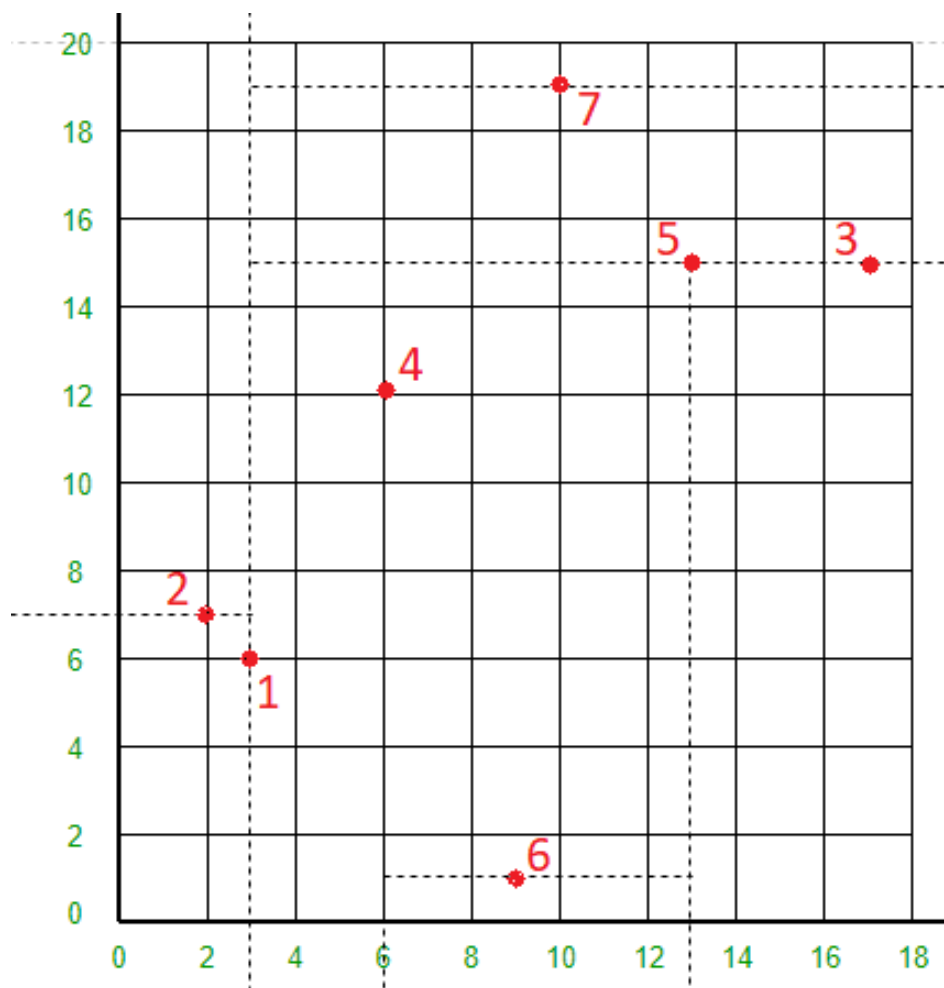


Figura 2: División del espacio en KD-Tree

## 4.2. KNN

### 4.2.1. Definición

K-Nearest Neighbors es uno de los algoritmos de clasificación más básicos pero esenciales en Machine Learning. Pertenecer al dominio de aprendizaje supervisado y encuentra una intensa aplicación en el reconocimiento de patrones, minería de datos y detección de intrusos. Es ampliamente disponible en escenarios de la vida real ya que no es paramétrico, lo que significa que no hace suposiciones subyacentes sobre la distribución de datos (a diferencia de otros algoritmos como GMM, que asume una distribución gaussiana de los datos dados). Nos dan unos datos previos (también llamados datos de entrenamiento), que clasifican las coordenadas en grupos identificados por un atributo. Como ejemplo, considere la figura 3 que muestra una tabla de puntos de datos que contienen dos características:

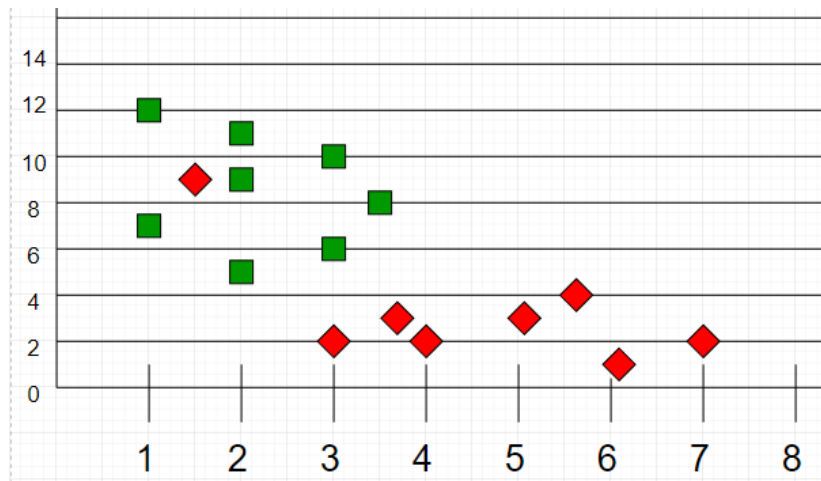


Figura 3: Datos de entrenamiento en KNN

Ahora, dado otro conjunto de puntos de datos (también llamados datos de prueba), asigne estos puntos a un grupo analizando el conjunto de entrenamiento. Tenga en cuenta que los puntos no clasificados están marcados como 'Blanco', de acuerdo a la figura 4

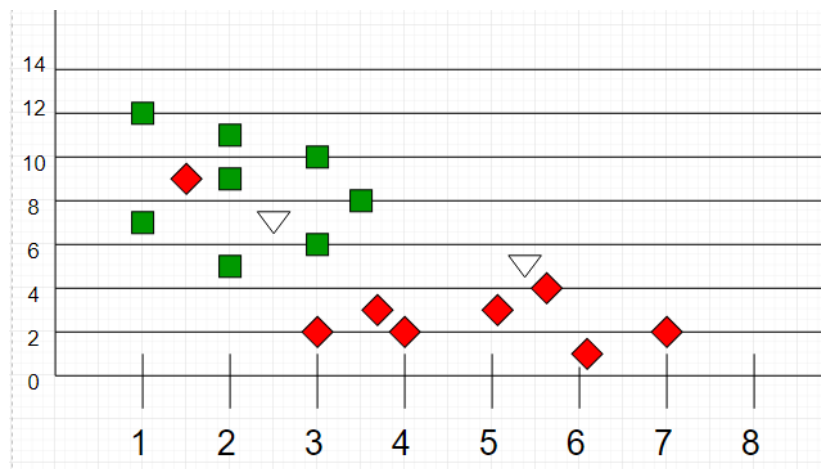


Figura 4: Datos de prueba en KNN

#### 4.2.2. Intuición

Si trazamos estos puntos en un gráfico, podemos ubicar algunos grupos o grupos. Ahora, dado un punto sin clasificar, podemos asignarlo a un grupo observando a qué grupo pertenecen sus vecinos más cercanos. Esto significa que un punto cercano a un grupo de puntos clasificados como 'Rojo' tiene una mayor probabilidad de ser clasificado como Rojo". Intuitivamente, podemos ver que el primer punto (2.5, 7) debe clasificarse como 'Verde' y el segundo punto (5.5, 4.5) debe clasificarse como 'Rojo'.

#### 4.2.3. Algoritmo

Sea  $m$  el número de muestras de datos de entrenamiento. Sea  $p$  un punto desconocido.

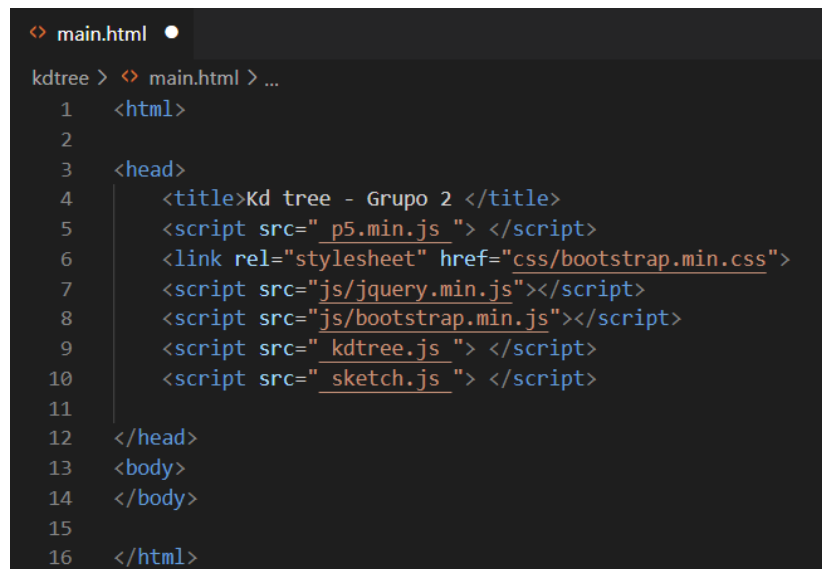
1. Almacene las muestras de entrenamiento en una matriz de puntos de datos  $arr[ ]$ . Esto significa que cada elemento de esta matriz representa una tupla  $(x, y)$ .
2. Haz el conjunto  $S$  de las  $K$  distancias más pequeñas obtenidas. Cada una de estas distancias corresponde a un punto de datos ya clasificado.
3. Devuelve la etiqueta mayoritaria entre  $S$ .

$K$  se puede mantener como un número impar para que podamos calcular una clara mayoría en el caso de que solo sean posibles dos grupos (por ejemplo, rojo/azul). Con el aumento de  $K$ , obtenemos límites más suaves y definidos a través de diferentes clasificaciones. Además, la precisión del clasificador anterior aumenta a medida que aumentamos la cantidad de puntos de datos en el conjunto de entrenamiento.

## 5. Ejercicios

### 5.1. Ejercicio N° 1

El la imagen 5 se presenta la creación del archivo html.



```
<> main.html ●
kdtree > <> main.html > ...
1  <html>
2
3  <head>
4      <title>Kd tree - Grupo 2 </title>
5      <script src="p5.min.js"></script>
6      <link rel="stylesheet" href="css/bootstrap.min.css">
7      <script src="js/jquery.min.js"></script>
8      <script src="js/bootstrap.min.js"></script>
9      <script src="kdtree.js"></script>
10     <script src="sketch.js"></script>
11
12 </head>
13 <body>
14 </body>
15
16 </html>
```

Figura 5: Archivo html

## 5.2. Ejercicio N° 2

- En la figura 6 se muestra la función **build\_kdtree**, que construye el KD-Tree y retorna el nodo raíz.

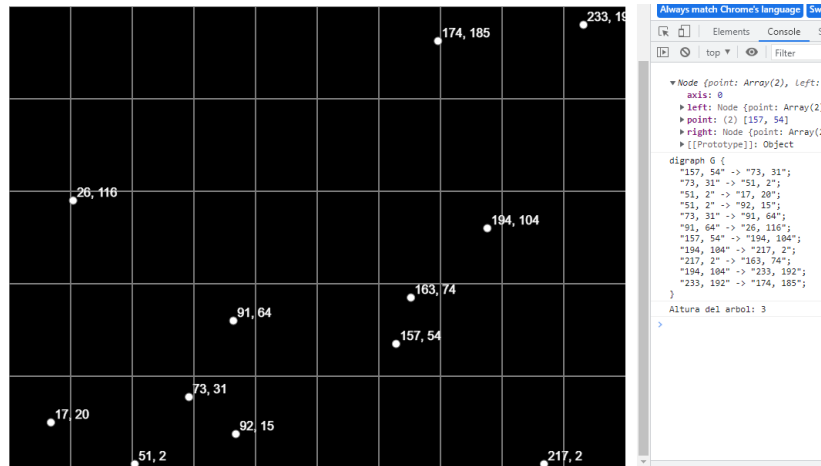


Figura 6: Nodo raíz en el KD-Tree

- En la figura 7 se muestra el resultado de la función **getHeight**, que retorna la altura de un árbol KD-Tree.
- En la figura 8 se muestra el resultado de la función **generate\_dot**, que genera al árbol KD-Tree en formato dot.

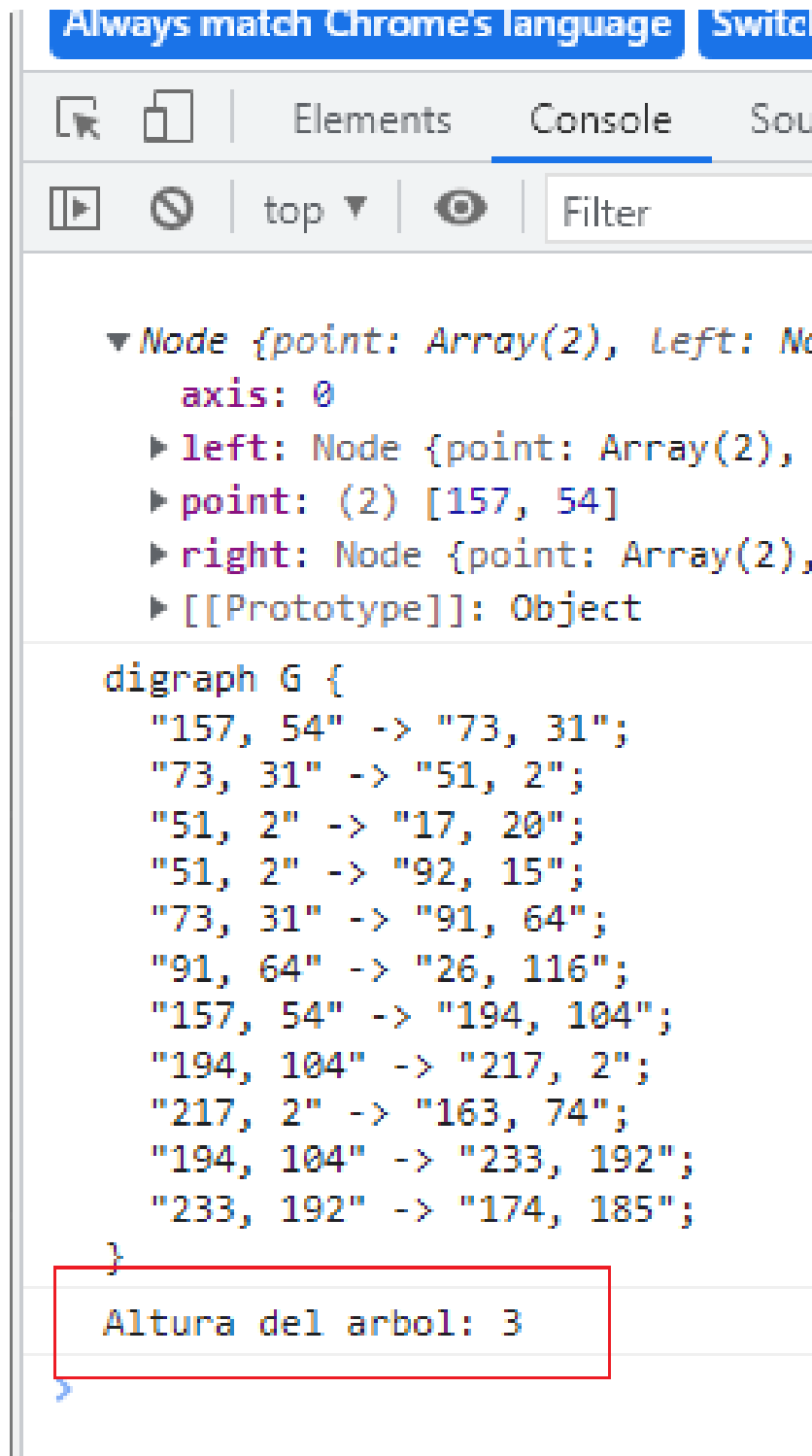


Figura 7: Altura del arbol KD-Tree



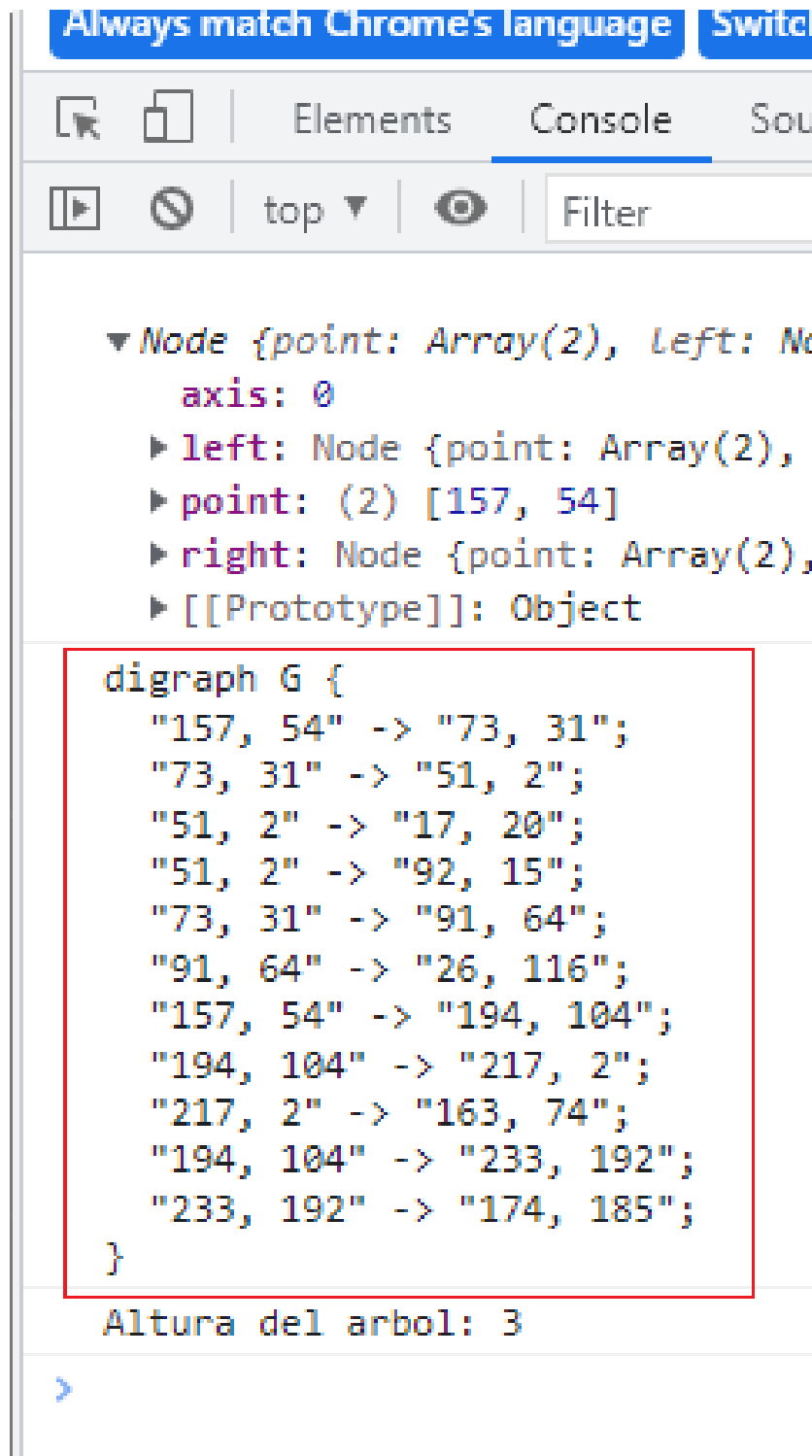


Figura 8: Altura del arbol KD-Tree

### 5.3. Ejercicio N° 3

El la imagen 9 se evalua los resultados después de crear el archivo *sketch.js*

```
cantidadK: 12 sketch.js:30
Node {point: Array(2), left: Node, right: Node, axis: 0} sketch.js:49
  axis: 0
  left: Node {point: Array(2), left: Node, right: Node, axis: 1}
  point: (2) [150, 18]
  right: Node {point: Array(2), left: Node, right: Node, axis: 1}
  [[Prototype]]: Object
digraph G { kdtree.js:243
  "150, 18" -> "126, 75";
  "126, 75" -> "77, 63";
  "77, 63" -> "28, 29";
  "77, 63" -> "120, 39";
  "126, 75" -> "50, 121";
  "50, 121" -> "45, 92";
  "150, 18" -> "156, 100";
  "156, 100" -> "235, 65";
  "235, 65" -> "159, 73";
  "156, 100" -> "248, 100";
  "248, 100" -> "187, 115";
}
Altura del arbol: 3 sketch.js:52
>
```

Figura 9: Evaluación de resultados en archivo *sketch.js*

## 5.4. Ejercicio N° 4

- En la figura 10 se muestra la implementación de la función *closest\_point\_force\_brute*.



Figura 10: Implementación de la función *closest\_point\_force\_brute*

- En la figura 11 se muestra la implementación de la función *naive\_closest*.

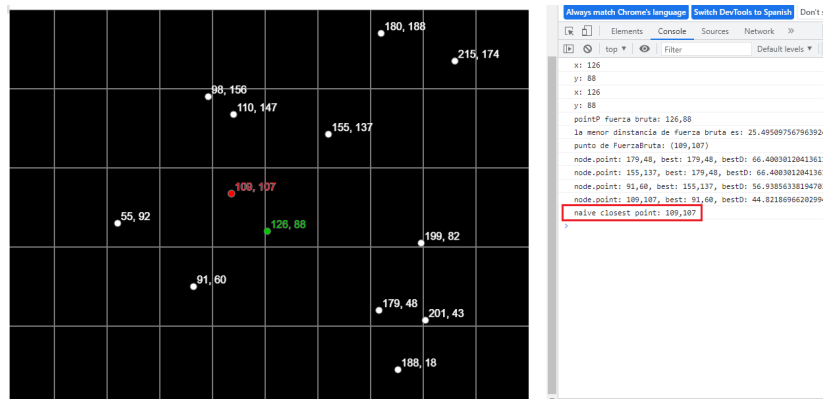


Figura 11: Implementación de la función *closest\_point\_force\_brute*

## 5.5. Ejercicio N° 5

- En la figura 12 se muestra evaluación de datos con la función *closest\_point\_force\_brute*.



Figura 12: Evaluación de datos con la función *closest\_point\_force\_brute*

- En la figura 13 se muestra evaluación de datos con la función *naive\_closest*.



Figura 13: Evaluación de datos con la función *naive\_closest*

## 5.6. Ejercicio N° 6

- En la figura 14 se muestra evaluación de datos con la función *closest\_point\_force\_brute*.

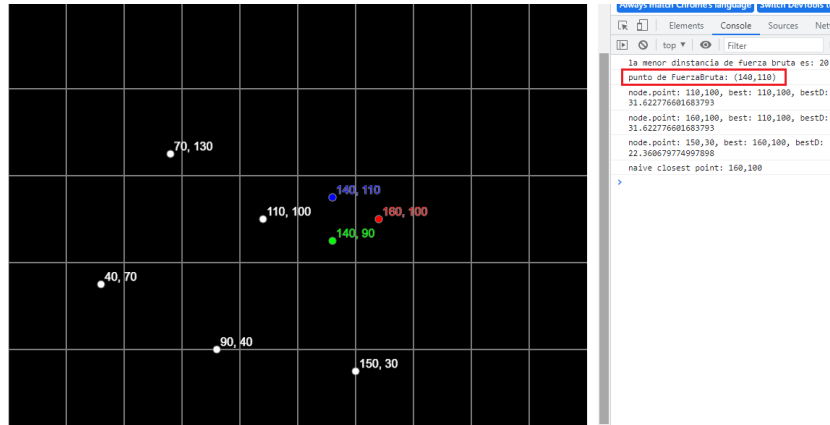


Figura 14: Evaluación de datos con la función *closest\_point\_force\_brute*

- En la figura 15 se muestra evaluación de datos con la función *naive\_closest*.

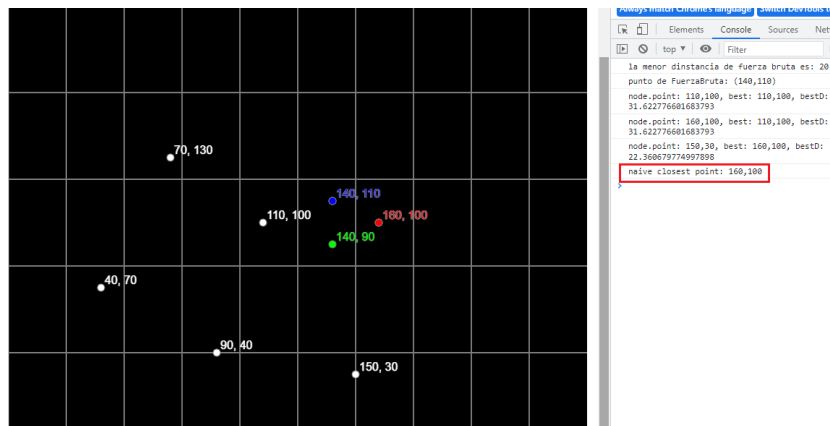


Figura 15: Evaluación de datos con la función *naive\_closest*

### 5.7. Ejercicio N° 7

El la figura 16 se muestra el resultado de la implementación de la función *closest\_point*.

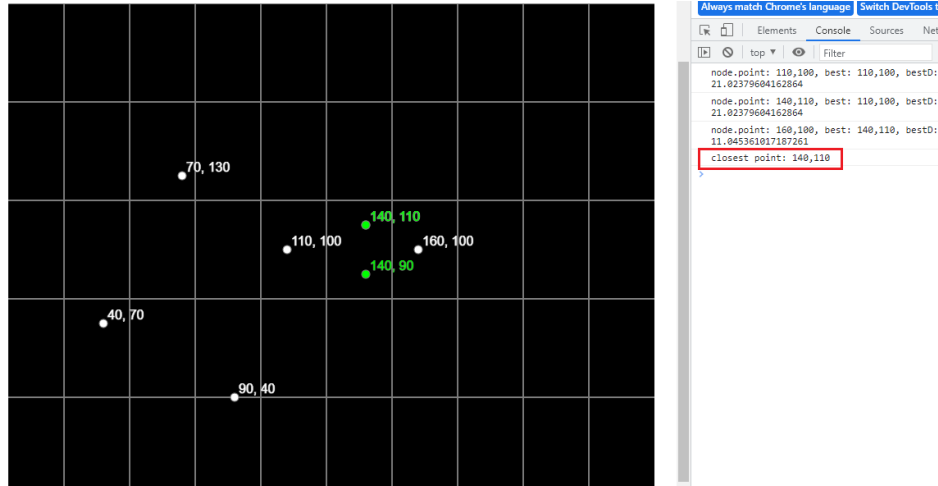


Figura 16: Función *closest\_point*

## 5.8. Ejercicio N° 8

El la figura 17 se muestra el resultado de la implementación de la función KNN, que retorna los k puntos mas cercanos a un punto.

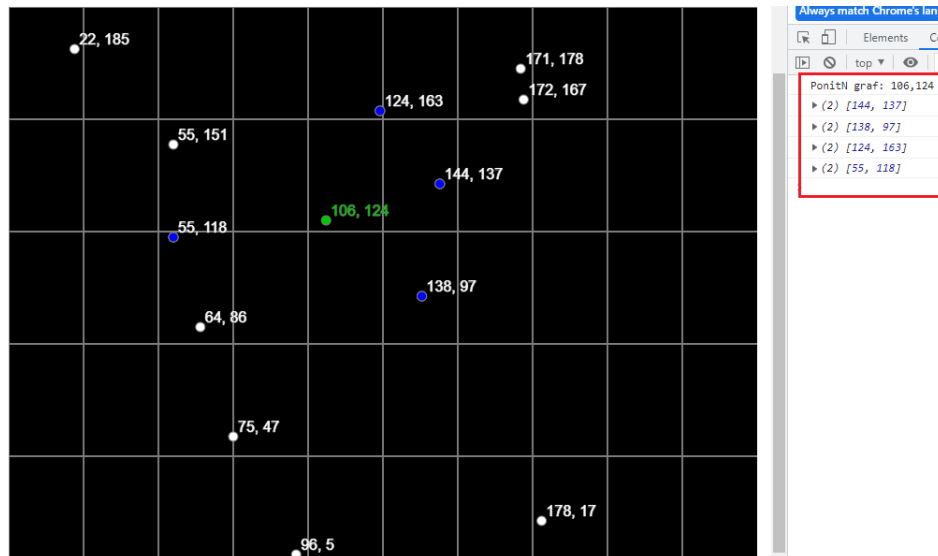


Figura 17: Función KNN, que retorna k puntos mas cercanos

### 5.9. Ejercicio N° 9

El la figura 18 se muestra el resultado de la implementación de la función *range\_query\_circle* de KD-Tree

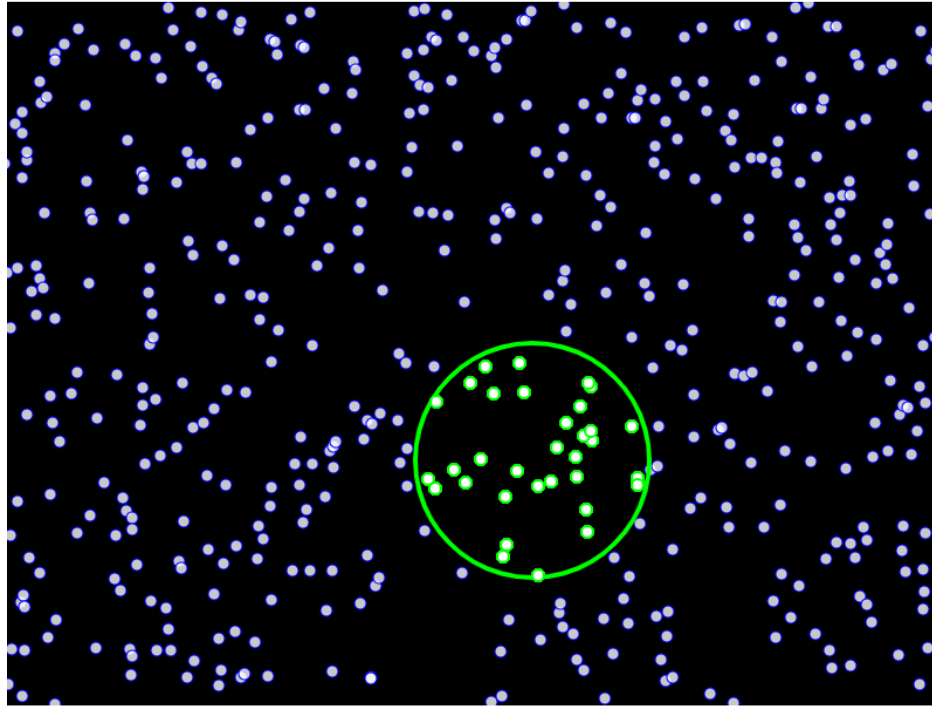


Figura 18: Función *range\_query\_circle* de KD-Tree



### 5.10. Ejercicio N° 10

El la figura 19 se muestra el resultado de la implementación de la función *range\_query\_rec* de KD-Tree

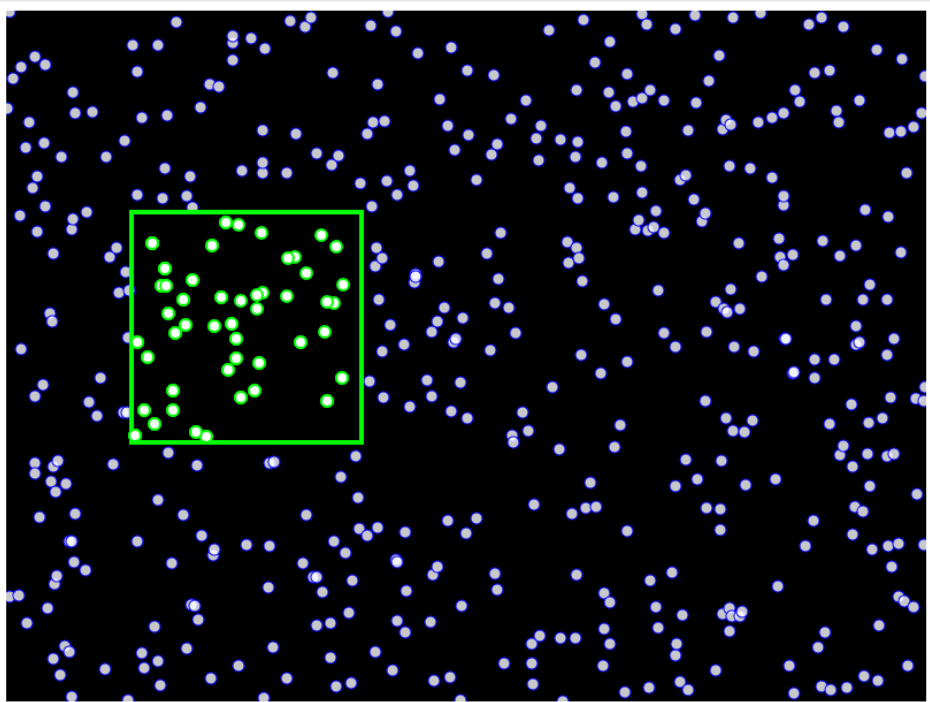


Figura 19: Función *range\_query\_rec* de KD-Tree

## 6. Conclusiones

- Los tiempos de ejecución de la función *closest\_point* es mas rápida que la función *closest\_point\_force\_brute*
- La función *naive\_closest* en algunas ocasiones no brinda el dato correcto.
- La función *closest\_point* se basa en la función *naive\_closest* pero haciendo mejoras para determinación del valor buscado de forma correcta.
- 
- De la bibliografía revisada, se concluye que las aplicaciones más comunes para los QuadTree y OcTree son el Procesamiento de imágenes, generación de mallas e indexado espacial.
- Las estructuras octree son usadas mayormente para partir un espacio tridimensional, dividiéndolo recursivamente en ocho octantes, siendo análogos tridimensionales de los quadtree bidimensionales.

## 7. Referencias

1. Amalia Duch, Vladimir Estivill-Castro, Conrado Martínez, “Randomized K-Dimensional Binary Search Trees”, M-RR/LSI-98-48-R, Universitat Politècnica de Catalunya, Barcelona, 1998.
2. C. Martínez y S. Roura, “Randomized binary search trees”, Journal of the ACM, Marzo 1998, Volumen 45, núm. 2, 288–323.
3. W. Cunto, G. Lau, y Ph. Flajolet, “Analysis of kd-trees: kd-trees improved by local reorganizations”, In F. Dehne, J. R. Sack y N. Santoro editors, Work, Algorithms and Data Structures, 1989, Volumen 382, 24–38.
4. Andrew W. Moore, “An Introductory tutorial on kd-trees”, Carnegie Mellon University, awm@cs.cmu.edu, 1992. <http://www.autonlab.org>.
5. Luc Devroye, Nicholas Broutin, Ketan Dalal y Erin McLeish, “The kdtreap”, Personal Communication.
6. P5.js
7. Geeksforgeeks