Práctica Final

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

PRÁCTICA	TEMA	DURACIÓN
Final	Trabajo final	3 horas

1. Datos de los estudiantes

1. **Grupo:** 09
2. **Integrantes:**
   - Asmat Fuentes, Franz Rogger
   - Esthela Espinoza, Fausto Danilo
   - Ojeda Mamani, Abel Eberth
   - Paredes Rodriguez, Raybert

2. Implementación KD Tree

1. **Introducción:** El presente trabajo pretende hacer una implementación del algoritmo KDTree y la búsqueda K-Nearest Neighbor (KNN), explora el concepto de usar una estructura de datos de árbol KD para identificar los colores cercanos en base a una selección de una paleta de colores dada.

   El código fuente de esta disponible en:

   `https://github.com/UNSA-MCC-2022/AyED_Trabajo_Final`

   Se pretende establecer una búsqueda de colores cercanos a uno seleccionado por el usaurio, retornando el nombre de esos colores cercanos en base a una paleta pre-establecida. En lugar de adivinar un color al azar, estamos tratando de limitar nuestras conjeturas a una lista seleccionada de colores. Un escenario de ejemplo sería analizar las fotos de productos de un catálogo de productos conocido con colores estándar.

   Se ha construido una paleta de colores en datos (JSON), en base a ello se realizarán las búsquedas, con este conjunto de datos se pretende realizar las búsquedas utilizando KNN, a continuación se muestra una porción de esta información:

Figura 1: Paleta de colores en datos

2. **Utilización:** Para el píxel del color seleccionado, elegiremos los color más similares de la paleta anterior. Básicamente, necesitamos convertir la el color en sus componentes RGB para que todos los colores resultado de la búsqueda sean de la paleta.

   Para ello, se necesita definir qué es la similitud de color. El enfoque más sencillo sería considerar el hecho de que cada color se compone de tres componentes: rojo, azul y verde, como un punto tridimensional. Si los representamos en un gráfico con valores de rojo, verde y azul como los tres ejes, podemos ver que los similares permanecen cerca uno del otro:

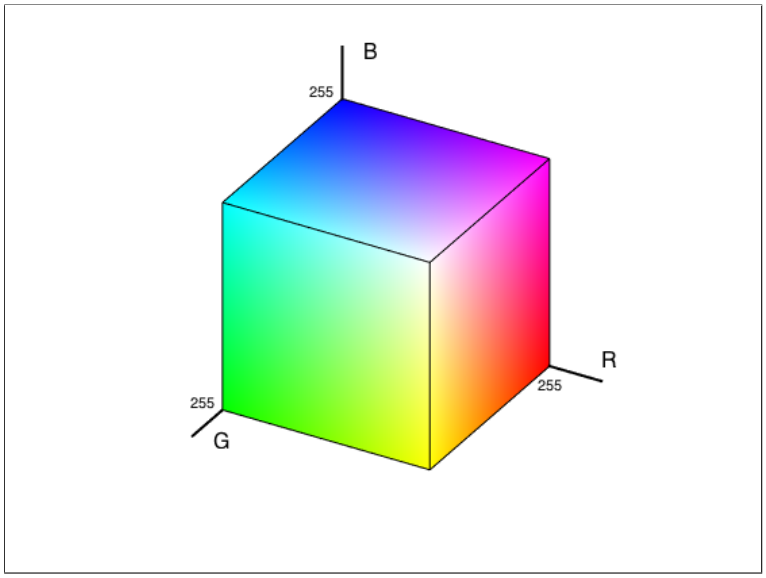

Figura 2: Representación 3D de los componentes RGB

Una vez definida la forma de descomponer el color en partes, lo siguiente es conocer que para encontar los colores similares, estos tienen los componentes RGB cercanos entre si. Es en ese pounto en que entra KNN, con este algorimo podemos encontrar puntos 3D cercanos entre si.

Para realizar esto se debe calcular la distancia entre los diferentes colores. En un sistema de coordenadas cartesianas 3D como el de la imagen, la ecuación de la distancia entre dos colores sería:

$$\sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

Donde:

$$(r_1,\ g_1\ y\ b_1)$$

Son del primer color y

$$(r_2,\ g_2\ y\ b_2)$$

del segundo color

Podemos encontrar la distancia entre el color de cada píxel contra cada uno de los colores de la paleta y encontrar el color de la paleta más similar.

3. **Implementación:** Para realizar las acciones anteriormente mencionadas se procede mostramos la implementación de KNN (en la función nearest del archivo kdtree.js)

```
// Algoritmo K-Nearest Neighbor(KNN)
function nearest (point, maxNodes, maxDistance) {
  var i,
    result,
    bestNodes;

  bestNodes = new BinaryHeap(
```

```
    function (e) { return -e[1]; }
  );

  function nearestSearch(node) {
    var bestChild,
      dimension = dimensions[node.dimension],
      ownDistance = metric(point, node.obj),
      linearPoint = {},
      linearDistance,
      otherChild,
      i;

    function saveNode(node, distance) {
      bestNodes.push([node, distance]);
      if (bestNodes.size() > maxNodes) {
        bestNodes.pop();
      }
    }

    for (i = 0; i < dimensions.length; i += 1) {
      if (i === node.dimension) {
        linearPoint[dimensions[i]] = point[dimensions[i]];
      } else {
        linearPoint[dimensions[i]] = node.obj[dimensions[i]];
      }
    }

    linearDistance = metric(linearPoint, node.obj);

    if (node.right === null && node.left === null) {
      if (bestNodes.size() < maxNodes || ownDistance < bestNodes.peek()[1]) {
        saveNode(node, ownDistance);
      }
      return;
    }

    if (node.right === null) {
      bestChild = node.left;
    } else if (node.left === null) {
      bestChild = node.right;
    } else {
      if (point[dimension] < node.obj[dimension]) {
        bestChild = node.left;
      } else {
        bestChild = node.right;
      }
    }

    nearestSearch(bestChild);

    if (bestNodes.size() < maxNodes || ownDistance < bestNodes.peek()[1]) {
      saveNode(node, ownDistance);
```

```
      }

      if (bestNodes.size() < maxNodes || Math.abs(linearDistance) < bestNodes.peek()[1]) {
        if (bestChild === node.left) {
          otherChild = node.right;
        } else {
          otherChild = node.left;
        }
        if (otherChild !== null) {
          nearestSearch(otherChild);
        }
      }
    }

    if (maxDistance) {
      for (i = 0; i < maxNodes; i += 1) {
        bestNodes.push([null, maxDistance]);
      }
    }

    if (self.root)
      nearestSearch(self.root);

    result = [];

    for (i = 0; i < Math.min(maxNodes, bestNodes.content.length); i += 1) {
      if (bestNodes.content[i][0]) {
        result.push([bestNodes.content[i][0].obj, bestNodes.content[i][1]]);
      }
    }
    return result;
  }
```

4. **Prueba en browser:** A continuación s epresenta la ejecución del codigo en browser, en el ejempo se puede apreciar que al elegir un color en el selector de colores, se muestran los diez (10) coolores mas cercanos a la paleta de colores predefinida.

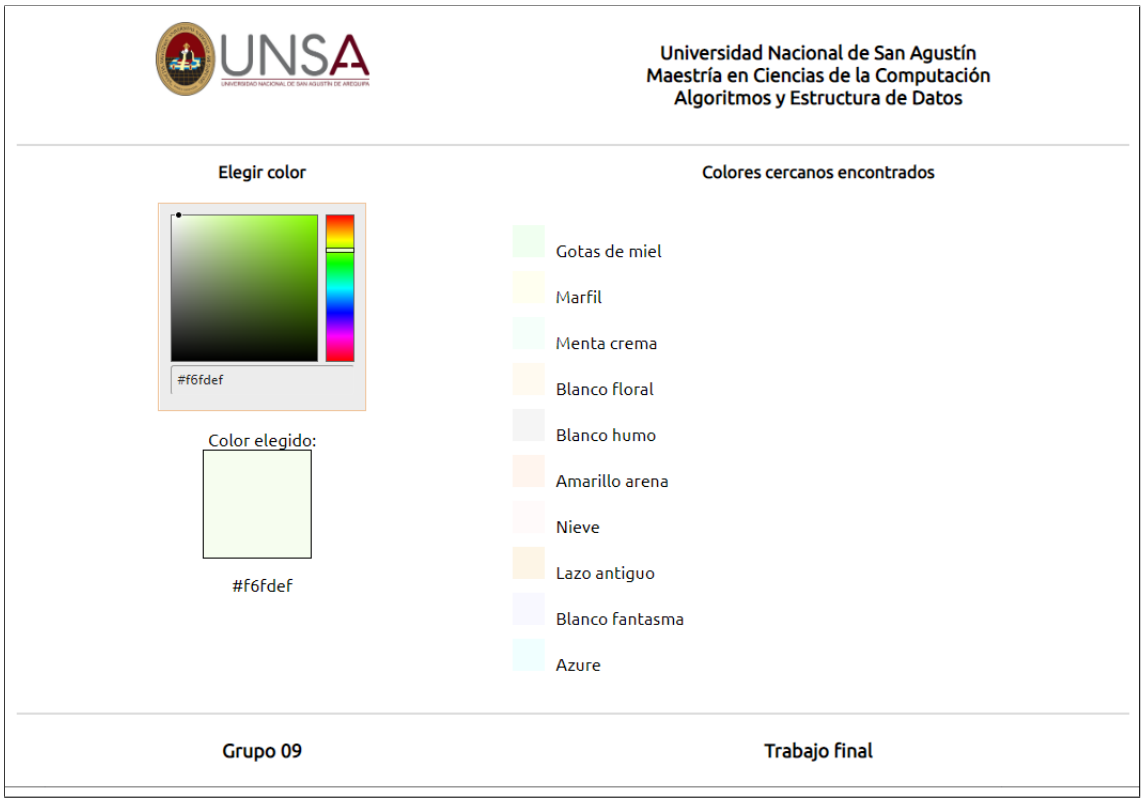

Figura 3: Busqueda de colores en ejecucion

5. **Conclusiones:** El método del árbol kd En pocas palabras, se crea una estructura de datos en forma de árbol para almacenar la paleta base. Esto preservará la información sobre la cercanía de cada color entre sí en la propia estructura de datos. Cuando se busca un color similar, en lugar de calcular la distancia a todos los colores de la paleta y se calculam solo a los pocos más cercanos, ahorrando el cálculo.

   En base a lo realizado, el metodo indicado puede ser usado para reconocimiento de colores dentro de imágenes convirtiendo la imagen en pixeles tratando de ubicar elementos dentro de la imagen, como por ejemplo:

Figura 4: Semaforo

   Una vez obtenida una paleta cercana a las imagenes que se queiren analizar, el siguiente paso sería agrupar los colores cercanos a cuando el semaforo esta encendido o apagado en cierta luz,

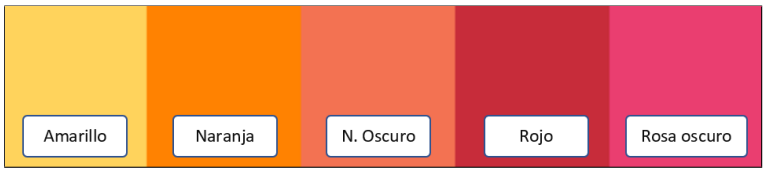

Figura 5: Paleta de color cercano al semaforo

Con los colores cercanos y agrupados hasta obtener cierta cereza que la luz del semaforo esta encendida o apagada

3. Repositorio

La implementación de los algoritmos y los datos utilizados es el siguiente:

`https://github.com/UNSA-MCC-2022/AyED_Trabajo_Final`

4. Representación gráfica

Se realizó la implementación de la representación gráfica de los algoritmos indicados, esto se pueden visualizar en el siguiente enlace:

`https://unsa-mcc-2022.github.io/AyED_Trabajo_Final/docs/index.html`