SEMANA 12 (junio 17, 19 ,21)

CONSULTAS DE PROXIMIDAD

Ejemplo de Problema: "Imagina que trabajas para una empresa de entrega de paquetes. Tienes un almacén central y necesitas saber cuál es el repartidor más cercano a una dirección específica para optimizar las entregas. ¿Cómo podrías determinar quién está más cerca de una dirección dada de manera eficiente?"

Problema:

- Determinar el repartidor más cercano a una dirección dada (consulta de vecino más cercano).
- Determinar los tres repartidores más cercanos a una dirección dada (consulta de k vecinos más cercanos).

Actividades:

- **Investigación sobre estructuras de datos:** Introducir conceptos básicos de kd-trees.
- **Estudio de métricas de distancia:** Explicar la distancia euclidiana y otras métricas.
- **Ejemplos prácticos:** Mostrar ejemplos de cómo las consultas de proximidad se utilizan en aplicaciones reales.

1. Aplicación Práctica

Objetivo: Hacer que los estudiantes implementen soluciones prácticas que resuelvan el problema planteado.

Actividades:

- **Construcción de kd-trees:**
    - Los estudiantes implementan una función para construir un kd-tree a partir de un conjunto de puntos (ubicaciones de repartidores).

```
class KDNode {
 constructor(point, axis, left = null, right = null) {
 this.point = point;
 this.axis = axis;
 this.left = left;
 this.right = right;
 }
}
```

```
function buildKDTree(points, depth = 0) {
 if (points.length === 0) {
 return null;
 }

 const axis = depth % 2;
 points.sort((a, b) => a[axis] - b[axis]);
 const medianIndex = Math.floor(points.length / 2);
 const medianPoint = points[medianIndex];

 return new KDNode(
 medianPoint,
 axis,
 buildKDTree(points.slice(0, medianIndex), depth + 1),
 buildKDTree(points.slice(medianIndex + 1), depth + 1)
 );
}
```

- **Implementación de consultas de proximidad:**
  - Los estudiantes implementan la búsqueda del vecino más cercano y la búsqueda de k vecinos más cercanos.

```
function distanceSquared(point1, point2) {
 return (point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2;
}

function nearestNeighbor(node, target, depth = 0, best = null) {
 if (node === null) {
 return best;
 }

 const axis = depth % 2;
 const nextBest = best === null || distanceSquared(target, node.point) <
distanceSquared(target, best.point) ? node : best;
 const nextDepth = depth + 1;

 let nextNode = null;
 let oppositeNode = null;

 if (target[axis] < node.point[axis]) {
 nextNode = node.left;
 oppositeNode = node.right;
 } else {
 nextNode = node.right;
 oppositeNode = node.left;
 }

 best = nearestNeighbor(nextNode, target, nextDepth, nextBest);
```

```
 if (distanceSquared(target, best.point) > (target[axis] - node.point[axis]) ** 2)
{
 best = nearestNeighbor(oppositeNode, target, nextDepth, best);
 }

 return best;
}

function kNearestNeighbors(node, target, k, depth = 0, heap = []) {
 if (node === null) {
 return heap;
 }

 const axis = depth % 2;
 const distance = distanceSquared(target, node.point);

 if (heap.length < k) {
 heap.push({ node: node, distance: distance });
 heap.sort((a, b) => a.distance - b.distance);
 } else if (distance < heap[heap.length - 1].distance) {
 heap[heap.length - 1] = { node: node, distance: distance });
 heap.sort((a, b) => a.distance - b.distance);
 }

 const nextNode = target[axis] < node.point[axis] ? node.left : node.right;
 const oppositeNode = target[axis] < node.point[axis] ? node.right : node.left;

 heap = kNearestNeighbors(nextNode, target, k, depth + 1, heap);

 if (heap.length < k || Math.abs(target[axis] - node.point[axis]) ** 2 <
heap[heap.length - 1].distance) {
 heap = kNearestNeighbors(oppositeNode, target, k, depth + 1, heap);
 }

 return heap;
}
```

2. Resolución del Problema y Discusión

Objetivo: Aplicar las soluciones implementadas para resolver el problema y discutir los resultados.

Actividades:

- **Ejecutar consultas de proximidad:**

- Los estudiantes prueban sus implementaciones con diferentes puntos de consulta para encontrar el repartidor más cercano y los tres repartidores más cercanos.

```
// Conjunto inicial de puntos (x, y)
const points = [
 [3, 6],
 [17, 15],
 [13, 15],
 [6, 12],
 [9, 1],
 [2, 7],
 [10, 19]
];

// Construir el kd-tree
const kdTree = buildKDTree(points);

// Punto de consulta
const queryPoint = [10, 5];

// Búsqueda de vecino más cercano
const nearest = nearestNeighbor(kdTree, queryPoint);
console.log(`Vecino más cercano a ${queryPoint}: (${nearest.point[0]},
${nearest.point[1]})`);

// Búsqueda de k vecinos más cercanos
const k = 3;
const kNearest = kNearestNeighbors(kdTree, queryPoint, k);
console.log(`Los ${k} vecinos más cercanos a ${queryPoint}:`);
kNearest.forEach((result, index) => {
 console.log(`${index + 1}: (${result.node.point[0]}, ${result.node.point[1]})`);
});
```

- **Discusión:**
  - Analizar los resultados obtenidos.
  - Comparar la eficiencia de las consultas de proximidad con otros métodos de búsqueda.
  - Reflexionar sobre cómo estas técnicas se aplican en situaciones del mundo real.