SEMANA 17 (Julio 22, 24 ,26)

FUNCIONES CONTINUAS

M-trees (Metric trees)

Los M-trees (Metric trees) son una estructura de datos especializada utilizada para organizar y gestionar colecciones de objetos en espacios métricos. Esta estructura es útil para realizar búsquedas eficientes, especialmente en grandes conjuntos de datos donde las distancias entre los elementos están definidas por una métrica. Se utiliza principalmente en aplicaciones que requieren la búsqueda de vecinos más cercanos, recuperación de información y reconocimiento de patrones en espacios métricos.

Propiedades y Características de los M-trees:

1. **Espacios Métricos:** Los M-trees están diseñados para trabajar en espacios métricos, donde la distancia entre dos puntos está definida por una función de distancia que satisface las propiedades de no negatividad, identidad, simetría y desigualdad triangular.
2. **Nodo:** Cada nodo en un M-tree puede contener varios elementos y se divide en:
    - Clústeres: Subconjuntos de elementos agrupados en un nodo.
    - Centroides: Cada clúster tiene un centroide o representante.
    - Rango de Cobertura: Cada nodo tiene un rango que define la distancia máxima desde el centroide a cualquier otro elemento en el clúster.
3. **Jerarquía:** Similar a los árboles B y B+, los M-trees son jerárquicos y equilibrados, con nodos internos que actúan como puntos de división y hojas que contienen los datos reales.
4. **División de Nodos:** Cuando un nodo excede su capacidad, se divide en dos nodos hijos, redistribuyendo los elementos y ajustando los centroides y rangos de cobertura.
5. **Búsqueda:** La búsqueda en un M-tree se realiza de manera eficiente navegando desde la raíz hacia las hojas, utilizando los centroides y los rangos de cobertura para descartar grandes porciones del espacio de búsqueda.

Aplicaciones de los M-trees:

1. **Recuperación de Información:** En sistemas de recuperación de información donde se necesita encontrar documentos o datos similares basados en alguna métrica de similitud.
2. **Reconocimiento de Patrones:** En sistemas de reconocimiento de patrones, como el reconocimiento de imágenes y sonidos, donde los datos se organizan en espacios métricos.

3. **Bioinformática:** Para gestionar y buscar en bases de datos de secuencias genéticas y estructuras moleculares.
4. **Minería de Datos:** En minería de datos para la agrupación y clasificación de grandes volúmenes de datos en espacios métricos.

Ejemplo de Implementación de M-tree:

Implementación de M-tree en JavaScript

A continuación, se presenta una implementación detallada de un M-tree en JavaScript, adaptada para ilustrar los conceptos en el contexto de funciones continuas en espacios métricos.

Clase `MTreeNode`

Define un nodo en el M-tree.

```
class MTreeNode {
  constructor(parent = null) {
    this.parent = parent;
    this.children = [];
    this.entries = [];
    this.centroid = null;
    this.coverageRadius = 0;
  }
}
```

Clase `MTree`

Define la estructura del M-tree y las operaciones básicas, como inserción y búsqueda.

```
class MTree {
  constructor(maxNodeSize = 4) {
    this.root = new MTreeNode();
    this.maxNodeSize = maxNodeSize;
  }

  _euclideanDistance(point1, point2) {
    let sum = 0;
    for (let i = 0; i < point1.length; i++) {
      sum += Math.pow(point1[i] - point2[i], 2);
    }
    return Math.sqrt(sum);
  }

  _calculateCentroid(entries) {
    let centroid = entries[0].coords.map(() => 0);
    for (let entry of entries) {
      for (let i = 0; i < entry.coords.length; i++) {
        centroid[i] += entry.coords[i];
      }
    }
```

```
    for (let i = 0; i < centroid.length; i++) {
      centroid[i] /= entries.length;
    }
    return centroid;
  }

  add(dataPoint) {
    let node = this.root;
    while (node.children.length > 0) {
      node = this._chooseSubtree(node, dataPoint.coords);
    }
    node.entries.push(dataPoint);
    if (node.entries.length > this.maxNodeSize) {
      this._splitNode(node);
    } else {
      node.centroid = this._calculateCentroid(node.entries);
    }
  }

  _chooseSubtree(node, dataPoint) {
    let minDistance = Infinity;
    let chosenSubtree = null;
    for (let child of node.children) {
      let distance = this._euclideanDistance(child.centroid, dataPoint);
      if (distance < minDistance) {
        minDistance = distance;
        chosenSubtree = child;
      }
    }
    return chosenSubtree;
  }

  _splitNode(node) {
    let midpoint = Math.floor(node.entries.length / 2);
    let leftNode = new MTreeNode(node.parent);
    let rightNode = new MTreeNode(node.parent);

    node.entries.sort((a, b) => a.coords[0] - b.coords[0]);

    leftNode.entries = node.entries.slice(0, midpoint);
    rightNode.entries = node.entries.slice(midpoint);

    leftNode.centroid = this._calculateCentroid(leftNode.entries);
    rightNode.centroid = this._calculateCentroid(rightNode.entries);

    node.children.push(leftNode, rightNode);
    node.entries = [];
    node.centroid = null;
  }

  search(queryPoint, range) {
    let results = [];
    this._searchRecursive(this.root, queryPoint, range, results);
    return results;
  }

  _searchRecursive(node, queryPoint, range, results) {
```

```
    for (let entry of node.entries) {
      if (this._euclideanDistance(entry.coords, queryPoint) <= range) {
        results.push(entry);
      }
    }
    for (let child of node.children) {
      if (this._euclideanDistance(child.centroid, queryPoint) -
child.coverageRadius <= range) {
        this._searchRecursive(child, queryPoint, range, results);
      }
    }
  }
}
```

Conclusión

Los M-trees son una herramienta poderosa para organizar y buscar datos en espacios métricos, especialmente cuando se trabaja con funciones continuas. Su estructura jerárquica y uso de métricas permite búsquedas eficientes y manejabilidad en aplicaciones del mundo real, como la búsqueda de puntos de interés en un mapa, la recuperación de información y el reconocimiento de patrones.

EJEMPLO PRACTICO

Vamos a formular un ejemplo práctico basado en un problema del mundo real usando M-trees. Este ejemplo será sobre la búsqueda de puntos de interés (POI) en una ciudad. Los puntos de interés pueden ser restaurantes, parques, tiendas, etc., y queremos encontrar aquellos que están cerca de una ubicación específica.

Problema

Supongamos que tenemos un conjunto de puntos de interés en una ciudad, cada uno con sus coordenadas geográficas (latitud y longitud). Queremos construir un M-tree para estos puntos de interés para poder realizar búsquedas eficientes de los POI más cercanos a una ubicación dada.

Solución

1. **Definir la estructura de los datos**: Cada punto de interés tiene una latitud, una longitud y un nombre.
2. **Construir el M-tree**: Insertar los puntos de interés en el M-tree.
3. **Realizar búsquedas**: Encontrar todos los puntos de interés dentro de un radio específico desde una ubicación dada.

Implementación en JavaScript

Paso 1: Definir las Clases Básicas

```
class MTreeNode {
  constructor(parent = null) {
    this.parent = parent;
    this.children = [];
    this.entries = [];
    this.centroid = null;
    this.coverageRadius = 0;
  }
}

class MTree {
  constructor(maxNodeSize = 4) {
    this.root = new MTreeNode();
    this.maxNodeSize = maxNodeSize;
  }

  _euclideanDistance(point1, point2) {
    let sum = 0;
    for (let i = 0; i < point1.length; i++) {
      sum += Math.pow(point1[i] - point2[i], 2);
    }
    return Math.sqrt(sum);
  }

  _calculateCentroid(entries) {
    let centroid = entries[0].coords.map(() => 0);
    for (let entry of entries) {
      for (let i = 0; i < entry.coords.length; i++) {
        centroid[i] += entry.coords[i];
      }
    }
    for (let i = 0; i < centroid.length; i++) {
      centroid[i] /= entries.length;
    }
    return centroid;
  }

  add(dataPoint) {
    let node = this.root;
    while (node.children.length > 0) {
      node = this._chooseSubtree(node, dataPoint.coords);
    }
    node.entries.push(dataPoint);
    if (node.entries.length > this.maxNodeSize) {
      this._splitNode(node);
    } else {
      node.centroid = this._calculateCentroid(node.entries);
    }
  }

  _chooseSubtree(node, dataPoint) {
    let minDistance = Infinity;
```

```
    let chosenSubtree = null;
    for (let child of node.children) {
      let distance = this._euclideanDistance(child.centroid, dataPoint);
      if (distance < minDistance) {
        minDistance = distance;
        chosenSubtree = child;
      }
    }
    return chosenSubtree;
  }

  _splitNode(node) {
    let midpoint = Math.floor(node.entries.length / 2);
    let leftNode = new MTreeNode(node.parent);
    let rightNode = new MTreeNode(node.parent);

    node.entries.sort((a, b) => a.coords[0] - b.coords[0]);

    leftNode.entries = node.entries.slice(0, midpoint);
    rightNode.entries = node.entries.slice(midpoint);

    leftNode.centroid = this._calculateCentroid(leftNode.entries);
    rightNode.centroid = this._calculateCentroid(rightNode.entries);

    node.children.push(leftNode, rightNode);
    node.entries = [];
    node.centroid = null;
  }

  search(queryPoint, range) {
    let results = [];
    this._searchRecursive(this.root, queryPoint, range, results);
    return results;
  }

  _searchRecursive(node, queryPoint, range, results) {
    for (let entry of node.entries) {
      if (this._euclideanDistance(entry.coords, queryPoint) <= range) {
        results.push(entry);
      }
    }
    for (let child of node.children) {
      if (this._euclideanDistance(child.centroid, queryPoint) -
child.coverageRadius <= range) {
        this._searchRecursive(child, queryPoint, range, results);
      }
    }
  }
}
```

Paso 2: Datos de los Puntos de Interés

```
const pointsOfInterest = [
  { name: "Restaurant A", coords: [40.7128, -74.0060] },
  { name: "Park B", coords: [40.7158, -74.0020] },
  { name: "Store C", coords: [40.7120, -74.0100] },
```

```
  { name: "Museum D", coords: [40.7200, -74.0000] },
  { name: "Restaurant E", coords: [40.7250, -74.0050] },
];
```

Paso 3: Construcción del M-tree

```
let mtree = new MTree(2);  // Usamos un tamaño de nodo pequeño para mostrar el
proceso de división más fácilmente

pointsOfInterest.forEach(poi => mtree.add(poi));
```

Paso 4: Búsqueda de Puntos de Interés Cercanos

```
let userLocation = [40.7130, -74.0070];  // Ubicación del usuario
let searchRadius = 0.01;  // Radio de búsqueda

let nearbyPOIs = mtree.search(userLocation, searchRadius);
console.log("Puntos de interés cercanos:", nearbyPOIs);
```

Explicación del Proceso

1. **Insertar Puntos de Interés**:
    - Insertamos cada punto de interés en el M-tree. Si el nodo actual excede su capacidad, se divide en dos nodos hijos.
2. **Buscar Puntos Cercanos**:
    - Usamos la ubicación del usuario y un radio de búsqueda para encontrar todos los puntos de interés dentro de ese rango. La búsqueda se realiza de manera eficiente gracias a la estructura jerárquica del M-tree.

Visualización del Estado del M-tree

Después de insertar los puntos de interés, el M-tree se dividirá de acuerdo con la capacidad máxima de los nodos y las distancias entre los puntos, optimizando la estructura para búsquedas eficientes.