SEMANA 13 (junio 24, 26 ,28)

ÁRBOLES PARA FUNCIONES DE DISTANCIAS DISCRETAS

Para estudiar árboles para funciones de distancias discretas, es esencial comprender varios conceptos clave. Estos conceptos proporcionan la base teórica y práctica necesaria para trabajar con este tipo de estructuras de datos. A continuación, se detallan los conceptos más importantes:

1. Espacio Métrico

Un espacio métrico es un conjunto de elementos donde se define una función de distancia que satisface ciertas propiedades. Para trabajar con árboles para distancias discretas, es fundamental entender estas propiedades:

- **No negatividad**: La distancia entre dos puntos es siempre no negativa.
- **Identidad**: La distancia entre un punto y sí mismo es cero.
- **Simetría**: La distancia entre dos puntos A y B es la misma que la distancia entre B y A.
- **Desigualdad triangular**: La distancia directa entre dos puntos es menor o igual que la suma de las distancias pasando por un tercer punto.

2. Funciones de Distancia Discreta

Una función de distancia discreta mide la "diferencia" entre dos elementos de un conjunto, produciendo un valor discreto (normalmente un número entero). Ejemplos comunes incluyen:

- **Distancia de Hamming**: Cuenta el número de posiciones en las que dos cadenas de igual longitud difieren.
- **Distancia de Levenshtein (o de edición)**: Mide el número mínimo de operaciones (inserciones, eliminaciones o sustituciones) necesarias para convertir una cadena en otra.

3. Árboles de Búsqueda Métrica

Un árbol de búsqueda métrica es una estructura de datos que organiza puntos en un espacio métrico para facilitar operaciones de búsqueda eficientes. Los tipos comunes incluyen:

- **Árbol BK (Burkhard-Keller Tree)**: Divide el espacio métrico utilizando distancias discretas desde un punto pivote, organizando los datos en una estructura de árbol.
- **Árboles Vantage Point (VP Trees)**: Utilizan puntos de vantage para dividir el espacio métrico en regiones, optimizando las búsquedas de proximidad.

4. Construcción del Árbol

La construcción del árbol implica seleccionar un punto pivote, calcular las distancias desde este punto a todos los demás puntos, y dividir los puntos en subgrupos basados en estas distancias. Este proceso es recursivo y se repite para cada subgrupo.

5. Operaciones de Búsqueda

Las operaciones de búsqueda en árboles para distancias discretas incluyen:

- **Búsqueda de vecinos más cercanos**: Encontrar los puntos más cercanos a un punto de consulta dentro de una distancia especificada.
- **Búsqueda de rango**: Encontrar todos los puntos dentro de un rango de distancia desde un punto de consulta.

6. Complejidad Computacional

Entender la complejidad computacional de las operaciones de construcción y búsqueda es crucial. La eficiencia de un árbol para distancias discretas depende de la estructura del árbol y de la función de distancia utilizada.

7. Aplicaciones Prácticas

Árboles para distancias discretas tienen muchas aplicaciones prácticas, incluyendo:

- **Reconocimiento de patrones**: Identificación de patrones similares en datos.
- **Biología computacional**: Análisis de secuencias genéticas.
- **Recuperación de información**: Búsqueda de documentos o elementos similares en bases de datos.
- **Visión por computadora**: Búsqueda de imágenes similares basadas en características métricas.

8. Implementación y Optimización

Es importante no solo entender la teoría, sino también ser capaz de implementar y optimizar estas estructuras en un lenguaje de programación. Familiarizarse con técnicas de optimización específicas para estas estructuras puede mejorar significativamente el rendimiento.

EJEMPLO PRÁCTICO DE ÁRBOLES PARA FUNCIONES DE DISTANCIAS DISCRETAS

Problema: "Supongamos que estás desarrollando una aplicación para corregir errores tipográficos en nombres de usuarios. Cuando un usuario introduce un nombre que no existe en la base de datos, queremos sugerir nombres similares. ¿Cómo podríamos estructurar y organizar nuestros datos para hacer esto de manera eficiente?"

Objetivo: Hacer que el estudiante identifique y discuta las dificultades y necesidades del problema.

Preguntas para discutir:

- ¿Qué significa que un nombre sea "similar" a otro?
- ¿Cómo podríamos medir la "similitud" entre dos nombres?
- ¿Qué estrategias conoces para encontrar elementos similares en una gran colección de datos?

Conceptos Clave

Espacios Métricos y Funciones de Distancia Discreta:

- Un espacio métrico es un conjunto de elementos con una función de distancia definida que mide qué tan "lejanos" están dos elementos (Véase 1. Espacios Métricos).
- La **distancia de Levenshtein** es una forma de medir la diferencia entre dos cadenas de texto (Véase 2. Funciones de Distancia Discreta).

Ejemplo de Código:

```
function levenshteinDistance(a, b) {
    const matrix = [];

    for (let i = 0; i <= b.length; i++) {
        matrix[i] = [i];
    }
    for (let j = 0; j <= a.length; j++) {
        matrix[0][j] = j;
    }

    for (let i = 1; i <= b.length; i++) {
        for (let j = 1; j <= a.length; j++) {
```

```
            if (b.charAt(i - 1) === a.charAt(j - 1)) {
                matrix[i][j] = matrix[i - 1][j - 1];
            } else {
                matrix[i][j] = Math.min(
                    matrix[i - 1][j - 1] + 1,
                    Math.min(matrix[i][j - 1] + 1, matrix[i - 1][j] + 1)
                );
            }
        }
    }

    return matrix[b.length][a.length];
}
```

Introducción a los Árboles para Distancias Discretas

Concepto del Árbol BK:

- Un Árbol BK es una estructura que nos permite organizar datos basados en distancias discretas (Véase 3. Árboles de Búsqueda Métrica).

Ejemplo de Construcción del Árbol BK:

```
class Node {
    constructor(point) {
        this.point = point;
        this.children = {};
    }
}

class BKTree {
    constructor(distanceFunc) {
        this.root = null;
        this.distanceFunc = distanceFunc;
    }

    add(point) {
        if (this.root === null) {
            this.root = new Node(point);
        } else {
            this._add(this.root, point);
        }
    }

    _add(node, point) {
        const dist = this.distanceFunc(node.point, point);
        if (node.children[dist] === undefined) {
            node.children[dist] = new Node(point);
        } else {
            this._add(node.children[dist], point);
        }
    }

    search(point, maxDistance) {
```

```
        const results = [];
        this._search(this.root, point, maxDistance, results);
        return results;
    }

    _search(node, point, maxDistance, results) {
        const dist = this.distanceFunc(node.point, point);
        if (dist <= maxDistance) {
            results.push({ point: node.point, distance: dist });
        }
        for (let d = Math.max(0, dist - maxDistance); d <= dist + maxDistance;
d++) {
            if (node.children[d] !== undefined) {
                this._search(node.children[d], point, maxDistance, results);
            }
        }
    }
}
```

Ejercicio Práctico

Construcción del Árbol BK y Búsqueda de Nombres Similares:

- Se tiene una lista de nombres con los que se debe construir un árbol BK.
- Se realizan busquedas de nombres similares para un nombre de usuario ingresado, usando la distancia de Levenshtein y el árbol BK.

```
// Lista de nombres de ejemplo
const names = ["Alice", "Bob", "Charlie", "David", "Eve"];

// Crear el árbol BK con la distancia de Levenshtein
const tree = new BKTree(levenshteinDistance);

// Agregar nombres al árbol
names.forEach(name => tree.add(name));

// Buscar nombres similares dentro de una distancia de Levenshtein de 2
const queryName = "Alic";
const maxDistance = 2;
const similarNames = tree.search(queryName, maxDistance);

// Mostrar resultados
console.log('Nombres similares:', similarNames);
```