SEMANA 16 (julio 15, 17 ,19)

FUNCIONES CONTÍNUAS

Las funciones continuas tienen una definición matemática y su aplicación específica puede variar dependiendo del uso en estructuras de datos métricas.

Definición Matemática de Funciones Continuas

Una función $f: X \to Y$ es continua si para cualquier punto $x_0 \in X$ y cualquier $\varepsilon > 0$, existe un $\delta > 0$ tal que para todos los puntos $x \in X$ que satisfacen $|x - x_0| < \delta$, tenemos $|f(x) - f(x_0)| < \varepsilon$.

Esto significa que pequeños cambios en la entrada de la función resultan en pequeños cambios en la salida.

Ejemplo Cotidiano: Temperatura a lo Largo del Día

Supongamos que tenemos una función $T(t)$ que representa la temperatura en grados Celsius a lo largo del tiempo t en horas durante un día.

Datos Reales (Ejemplo):

Aquí hay algunas temperaturas registradas en diferentes horas de un día:

- 6:00 AM: 15°C
- 9:00 AM: 18°C
- 12:00 PM: 22°C
- 3:00 PM: 24°C
- 6:00 PM: 20°C
- 9:00 PM: 17°C

Queremos verificar si la función que describe esta relación es continua.

1. **Explicación Visual:**

Imaginemos que tenemos una gráfica donde el eje x representa el tiempo en horas y el eje y representa la temperatura en grados Celsius.

1. **Observación Local (Punto t_0)**:
   - Consideremos el punto $t_0 = 12PM$ (mediodía).
   - La temperatura $T(t_0) = 22$℃.
2. **Condición de Continuidad**:
   - Para cualquier $\varepsilon > 0$, queremos encontrar un $\delta > 0$ tal que si $|t - 12| < \delta$, entonces $|T(t) - 22| < \varepsilon$.

Verificación:

Supongamos que queremos que los cambios en la temperatura no sean mayores a 1°C, es decir, $\varepsilon = 1$.

- Si nos movemos una hora antes o después de las 12 PM, las temperaturas son:
  - A las 11:00 AM, supongamos que $T(11) = 21$°C.
  - A la 1:00 PM, supongamos que $T(13) = 23$°C.
- Observamos que:
  - $|T(11) - 22| = |21 - 22| = 1$ (menor o igual que ε).
  - $|T(13) - 22| = |23 - 22| = 1$ (menor o igual que ε).

En este caso, podemos decir que para $\delta = 1$ hora, si $|t - 12| < 1$ hora, entonces $|T(t) - 22| < 1$ grado Celsius. Esto cumple la definición de continuidad.

Ilustración Gráfica

Para visualizar mejor este ejemplo, podríamos graficar la función:

```
import matplotlib.pyplot as plt

# Datos de temperatura a lo largo del día
horas = [6, 9, 12, 15, 18, 21]
temperaturas = [15, 18, 22, 24, 20, 17]

plt.plot(horas, temperaturas, marker='o')
plt.xlabel('Hora del día')
plt.ylabel('Temperatura (°C)')
plt.title('Temperatura a lo largo del día')
plt.grid(True)
plt.show()
```

Conclusión

La función $T(t)$ que representa la temperatura a lo largo del día es continua si para cualquier punto en el tiempo, pequeños cambios en el tiempo resultan en pequeños cambios en la temperatura. Este ejemplo con datos reales demuestra cómo se aplica la definición matemática de funciones continuas en un contexto cotidiano.

Aplicación en Métodos de Acceso Métrico

En el contexto de estructuras de datos métricas, como las que se utilizan para organizar y buscar datos en espacios métricos, las funciones continuas pueden tener varias aplicaciones:

1. **Espacios Métricos y Continuidad:**

- En un espacio métrico, se define una función de distancia $d: X \times X \rightarrow \mathbb{R}$ que cumple con ciertas propiedades (no negatividad, identidad del discernible, simetría, desigualdad triangular). Una función continua en este contexto garantiza que la distancia entre puntos cercanos también sea pequeña.

2. **Métricas Continuas:**

   - Una métrica continua asegura que las pequeñas variaciones en los puntos del espacio no causen grandes cambios en la distancia calculada. Esto es crucial para estructuras de datos que dependen de la proximidad, como árboles métricos (M-trees) o estructuras de índices métricos.

3. **Funciones de Similaridad:**

   - En métodos de acceso métrico, las funciones de similaridad (que pueden ser vistas como funciones continuas) son utilizadas para determinar qué tan "cerca" o "parecidos" son dos elementos. Una función de similaridad continua asegura transiciones suaves en la evaluación de proximidad entre elementos.

Ejemplo: Uso en Árboles Métricos (M-trees)

Un árbol métrico (M-tree) es una estructura de datos utilizada para organizar elementos en un espacio métrico de tal manera que se faciliten las operaciones de búsqueda de proximidad (como búsquedas de rango o búsquedas de vecinos más cercanos).

- **Continuidad de la Métrica:** Si la métrica utilizada en el M-tree es continua, esto garantiza que para cualquier nodo del árbol, los elementos almacenados en sus subnodos son similares en términos de la métrica. Esto es importante para la eficiencia de las búsquedas, ya que permite que la estructura del árbol refleje de manera precisa las relaciones de proximidad entre los datos.

Ejemplo Práctico

Imaginemos que tenemos un espacio métrico donde los puntos representan imágenes y la métrica es una función de distancia basada en características visuales (como el histograma de color). Una función continua en este contexto asegura que imágenes similares (con características visuales cercanas) tendrán distancias pequeñas.

```
# Ejemplo de una función continua en un espacio métrico

def distancia_histograma(imagen1, imagen2):
    # Calcula la distancia entre dos imágenes basándose en sus histogramas
    hist1 = calcular_histograma(imagen1)
```

```
    hist2 = calcular_histograma(imagen2)
    return sum((h1 - h2) ** 2 for h1, h2 in zip(hist1, hist2)) ** 0.5

def calcular_histograma(imagen):
    # Esta función calcula el histograma de la imagen
    # Para simplificar, asumimos que retorna una lista de valores
    return [0] * 256  # Ejemplo simplificado
```

En este ejemplo, distancia_histograma es una función continua en el sentido de que pequeñas variaciones en los histogramas (debido a pequeños cambios en las imágenes) resultarán en pequeñas variaciones en la distancia calculada.

Resumen

Las funciones continuas, en el contexto de métodos de acceso métricos en estructuras de datos avanzadas, aseguran que las variaciones suaves en los datos no produzcan cambios abruptos en la evaluación de la distancia o similaridad. Esto es crucial para el rendimiento y precisión de las estructuras de datos métricas utilizadas para búsquedas y organización eficiente de los datos.

PROBLEMA PRÁCTICO

Problema: Supongamos que estamos desarrollando un sistema para una biblioteca. Queremos implementar una funcionalidad que permita buscar libros por su ID de manera eficiente. Cada libro tiene un ID único, un título y un autor. Queremos poder insertar nuevos libros, buscar libros por su ID y eliminar libros de la biblioteca.

Solución con Binary Search Trees (BSTs)

Vamos a utilizar un BST para almacenar y gestionar los libros. La estructura del BST nos permitirá realizar operaciones de búsqueda, inserción y eliminación de manera eficiente.

Implementación en JavaScript

Paso 1: Definir la estructura del nodo del árbol

Cada nodo en el BST representará un libro, con su ID, título y autor.

```
class Node {
  constructor(id, title, author) {
    this.id = id;
    this.title = title;
    this.author = author;
```

```
    this.left = null;
    this.right = null;
  }
}
```

Paso 2: Implementar el Binary Search Tree

Implementaremos las operaciones básicas: inserción, búsqueda y eliminación.

```
class BinarySearchTree {
  constructor() {
    this.root = null;
  }

  // Método para insertar un nuevo libro
  insert(id, title, author) {
    const newNode = new Node(id, title, author);
    if (this.root === null) {
      this.root = newNode;
    } else {
      this._insertNode(this.root, newNode);
    }
  }

  _insertNode(node, newNode) {
    if (newNode.id < node.id) {
      if (node.left === null) {
        node.left = newNode;
      } else {
        this._insertNode(node.left, newNode);
      }
    } else {
      if (node.right === null) {
        node.right = newNode;
      } else {
        this._insertNode(node.right, newNode);
      }
    }
  }

  // Método para buscar un libro por ID
  search(id) {
    return this._searchNode(this.root, id);
  }

  _searchNode(node, id) {
    if (node === null) {
      return null;
    }
    if (id < node.id) {
      return this._searchNode(node.left, id);
    } else if (id > node.id) {
```

```
      return this._searchNode(node.right, id);
    } else {
      return node;
    }
  }

  // Método para eliminar un libro por ID
  remove(id) {
    this.root = this._removeNode(this.root, id);
  }

  _removeNode(node, id) {
    if (node === null) {
      return null;
    }
    if (id < node.id) {
      node.left = this._removeNode(node.left, id);
      return node;
    } else if (id > node.id) {
      node.right = this._removeNode(node.right, id);
      return node;
    } else {
      // Nodo encontrado
      if (node.left === null && node.right === null) {
        node = null;
        return node;
      }
      if (node.left === null) {
        node = node.right;
        return node;
      } else if (node.right === null) {
        node = node.left;
        return node;
      }

      const aux = this._findMinNode(node.right);
      node.id = aux.id;
      node.title = aux.title;
      node.author = aux.author;
      node.right = this._removeNode(node.right, aux.id);
      return node;
    }
  }

  _findMinNode(node) {
    if (node.left === null) {
      return node;
    } else {
      return this._findMinNode(node.left);
    }
  }
```

```
  // Método para realizar recorrido inorden (para ver todos los libros ordenados
por ID)
  inorderTraversal() {
    this._inorderTraversalNode(this.root);
  }

  _inorderTraversalNode(node) {
    if (node !== null) {
      this._inorderTraversalNode(node.left);
      console.log(`ID: ${node.id}, Title: ${node.title}, Author:
${node.author}`);
      this._inorderTraversalNode(node.right);
    }
  }
}
```

Ejemplo de Uso

Vamos a utilizar nuestra clase `BinarySearchTree` para manejar una colección de libros en la biblioteca.

```
const bst = new BinarySearchTree();

// Insertar algunos libros
bst.insert(3, 'El Quijote', 'Miguel de Cervantes');
bst.insert(1, 'Cien años de soledad', 'Gabriel García Márquez');
bst.insert(4, 'Don Juan Tenorio', 'José Zorrilla');
bst.insert(2, 'La casa de Bernarda Alba', 'Federico García Lorca');

// Buscar un libro por ID
const book = bst.search(2);
if (book !== null) {
  console.log(`Libro encontrado: ID: ${book.id}, Título: ${book.title}, Autor:
${book.author}`);
} else {
  console.log('Libro no encontrado');
}

// Eliminar un libro por ID
bst.remove(3);

// Mostrar todos los libros en orden
console.log('Recorrido inorden de todos los libros:');
bst.inorderTraversal();
```

Explicación

4. Definición del Problema:

- Tenemos una biblioteca que necesita gestionar sus libros eficientemente, permitiendo inserciones, búsquedas y eliminaciones rápidas.

5. **Concepto de BST:**

   - Explicamos cómo un BST nos ayuda a realizar estas operaciones de manera eficiente al mantener los libros ordenados por sus IDs.

6. **Implementación en JavaScript:**

   - Paso a paso, implementamos los nodos y el árbol, mostrando cómo insertar, buscar y eliminar elementos en un BST.

7. **Ejemplo de Uso:**

   - Proveemos un ejemplo claro y completo de cómo utilizar la clase `BinarySearchTree` para gestionar los libros en la biblioteca.

Este enfoque permite a los estudiantes ver cómo un concepto teórico, como los Binary Search Trees, se puede aplicar para resolver problemas reales de manera eficiente.