SEMANA 14 (julio 01, 03 ,05)

MÉTODOS BASADOS EN HASH MULTIDIMENSIONAL: Métodos basados en Grids (Estructura de datos para acceso espacial)

1. Estructuras de Datos Espaciales

- **Cuadrículas (Grids)**: Es una estructura de datos espacial que divide el espacio en celdas uniformes. Cada celda puede ser tratada como una entidad separada, lo que facilita la localización de puntos dentro de ese espacio.

Concepto: La ciudad se divide en celdas uniformes, cada una representando un área específica.

Ejemplo: Supongamos que tenemos una ciudad con coordenadas que varían de (40.7120, -74.0060) a (40.7150, -74.0020). Dividimos esta área en celdas de 0.001 grados de tamaño.

```
// Crear una cuadrícula con tamaño de celda 0.001
const grid = new Grid(0.001);

// Añadir restaurantes a la cuadrícula
const restaurants = [
    new Restaurant("Pizza Place", 40.7125, -74.0055),
    new Restaurant("Burger Joint", 40.7130, -74.0045),
    new Restaurant("Sushi Bar", 40.7140, -74.0035)
];

restaurants.forEach(restaurant => grid.addRestaurant(restaurant));
```

2. Hash Multidimensional

- **Hashing**: Es el proceso de transformar una entrada (o "clave") en una salida de longitud fija, generalmente mediante una función de hash. En el contexto espacial, las coordenadas (latitud y longitud) se transforman en claves que corresponden a celdas específicas de una cuadrícula.

Concepto: Transformar coordenadas en claves de celdas.

Ejemplo: Convertimos coordenadas (40.7125, -74.0055) en claves.

```
const restaurant = new Restaurant("Pizza Place", 40.7125, -74.0055);
const key = grid.getCellKey(restaurant.latitude, restaurant.longitude);
console.log(key);  // Output: "40712,-74005"
```

- **Hash Multidimensional**: Adaptación del hashing para manejar múltiples dimensiones, como la latitud y longitud en nuestro ejemplo. Permite mapear coordenadas geográficas a una estructura de datos espaciales.

3. Geometría Computacional

- **Coordenadas Geográficas**: Sistema de referencia que utiliza latitud y longitud para definir la ubicación de puntos en la superficie de la Tierra.
- **División Espacial**: Técnica para dividir el espacio geográfico en áreas manejables, como las celdas en una cuadrícula, para facilitar la búsqueda y organización de datos.

Concepto: Usar latitud y longitud para definir ubicaciones.

Ejemplo: Representamos restaurantes con coordenadas geográficas.

```
const restaurant1 = new Restaurant("Pizza Place", 40.7125, -74.0055);
const restaurant2 = new Restaurant("Burger Joint", 40.7130, -74.0045);
console.log(restaurant1, restaurant2);
```

4. Algoritmos de Búsqueda

- **Búsqueda en Vecindad**: Técnica para encontrar todos los elementos dentro de una región específica alrededor de un punto de interés. En nuestro ejemplo, implica buscar en la celda correspondiente y en las celdas adyacentes para encontrar restaurantes cercanos.
- **Búsqueda Espacial Eficiente**: Métodos y técnicas para realizar búsquedas rápidas en grandes conjuntos de datos espaciales, minimizando el número de comparaciones necesarias.

Concepto: Buscar en celdas vecinas para encontrar elementos cercanos.

Ejemplo: Buscamos restaurantes cerca de una ubicación específica.

```
const userLocation = { latitude: 40.7128, longitude: -74.0050 };
const nearbyRestaurants = grid.getNearbyRestaurants(userLocation.latitude,
userLocation.longitude);
console.log(nearbyRestaurants);  // Output: Lista de restaurantes cercanos
```

5. Teoría de Complejidad

- **Complejidad Temporal**: Evaluación de cuánto tiempo toma ejecutar un algoritmo. En nuestro caso, comparar la eficiencia de la búsqueda en una estructura de cuadrícula frente a una búsqueda lineal en todos los datos.

- **Complejidad Espacial**: Evaluación de cuánto espacio de memoria utiliza una estructura de datos o un algoritmo. La estructura de cuadrícula puede ser más eficiente en términos de memoria para ciertos tipos de búsquedas espaciales.

Concepto: Evaluar cuánto tiempo toma ejecutar una búsqueda.

Ejemplo: Comparar la búsqueda en cuadrícula vs búsqueda lineal.

```
console.time("Linear                                                    Search");
const          linearResults          =          restaurants.filter(r          =>
    Math.abs(r.latitude      -      userLocation.latitude)      <      0.001      &&
    Math.abs(r.longitude       -       userLocation.longitude)       <       0.001);
console.timeEnd("Linear                                                 Search");

console.time("Grid                                                      Search");
const      gridResults      =      grid.getNearbyRestaurants(userLocation.latitude,
userLocation.longitude);
console.timeEnd("Grid Search");
```

6. Estrategias de Indexación

- **Indexación Espacial**: Métodos para organizar datos espaciales de manera que las consultas puedan ser respondidas eficientemente. La cuadrícula es una forma de indexación espacial que permite accesos rápidos a regiones específicas del espacio.
- **Buckets (Cubos)**: En la context de grids, un bucket es una celda de la cuadrícula que contiene los elementos que caen dentro de sus límites espaciales. Este concepto es esencial para entender cómo se agrupan y acceden los datos.

Concepto: Organizar datos espaciales para consultas eficientes.

Ejemplo: Usar celdas de cuadrícula para indexar restaurantes.

```
// Añadir restaurantes a celdas correspondientes
restaurants.forEach(restaurant => grid.addRestaurant(restaurant));

// Obtener restaurantes en una celda específica
const cellRestaurants = grid.getRestaurantsInCell(40.7125, -74.0055);
console.log(cellRestaurants);  // Output: Restaurantes en la celda
```

7. Conceptos de Programación

- **Clases y Objetos**: Entender cómo utilizar clases para modelar entidades del mundo real, como restaurantes y celdas de una cuadrícula.

- **Funciones de Hash**: Implementar funciones que convierten coordenadas geográficas en claves únicas para acceso rápido.
- **Estructuras de Datos en Programación**: Uso de Mapas (Maps) para almacenar y acceder a los datos de las celdas de la cuadrícula eficientemente.

Concepto: Modelar entidades del mundo real.

Ejemplo: Crear clases para restaurantes y celdas de cuadrícula.

```
class Restaurant {
    constructor(name, latitude, longitude) {
        this.name = name;
        this.latitude = latitude;
        this.longitude = longitude;
    }
}

class Grid {
    constructor(size) {
        this.size = size;
        this.cells = new Map();
    }

    getCellKey(latitude, longitude) {
        const x = Math.floor(latitude / this.size);
        const y = Math.floor(longitude / this.size);
        return `${x},${y}`;
    }

    addRestaurant(restaurant) {
        const key = this.getCellKey(restaurant.latitude, restaurant.longitude);
        if (!this.cells.has(key)) {
            this.cells.set(key, []);
        }
        this.cells.get(key).push(restaurant);
    }

    getRestaurantsInCell(latitude, longitude) {
        const key = this.getCellKey(latitude, longitude);
        return this.cells.get(key) || [];
    }

    getNearbyRestaurants(latitude, longitude) {
        const x = Math.floor(latitude / this.size);
        const y = Math.floor(longitude / this.size);
        const nearbyRestaurants = [];
        for (let dx = -1; dx <= 1; dx++) {
            for (let dy = -1; dy <= 1; dy++) {
```

```
                const key = `${x + dx},${y + dy}`;
                if (this.cells.has(key)) {
                    nearbyRestaurants.push(...this.cells.get(key));
                }
            }
        }
        return nearbyRestaurants;
    }
}
```

Aplicación de los Conceptos

8. Diseño de la Cuadrícula:

- Dividir la ciudad en celdas cuadradas de tamaño uniforme.
- Mapear coordenadas geográficas a celdas específicas usando una función de hash.

Ejemplo: Dividir la ciudad en celdas y mapear coordenadas.

```
const grid = new Grid(0.001);
const restaurants = [
    new Restaurant("Pizza Place", 40.7125, -74.0055),
    new Restaurant("Burger Joint", 40.7130, -74.0045),
    new Restaurant("Sushi Bar", 40.7140, -74.0035)
];

restaurants.forEach(restaurant => grid.addRestaurant(restaurant));
```

9. Inserción de Datos:

- Añadir restaurantes a las celdas correspondientes basándose en sus coordenadas geográficas.
- Utilizar un mapa para almacenar listas de restaurantes en cada celda.

Ejemplo: Añadir restaurantes a las celdas correspondientes.

```
restaurants.forEach(restaurant => grid.addRestaurant(restaurant));
```

10. Búsqueda Eficiente:

- Implementar búsqueda en la celda correspondiente a la ubicación del usuario y en las celdas adyacentes para encontrar todos los restaurantes cercanos.
- Comparar la eficiencia de esta búsqueda en comparación con una búsqueda lineal a través de todos los datos.

Ejemplo: Implementar búsqueda en celdas vecinas.

```
const userLocation = { latitude: 40.7128, longitude: -74.0050 };
const nearbyRestaurants = grid.getNearbyRestaurants(userLocation.latitude,
userLocation.longitude);
console.log(nearbyRestaurants);
```

11. Conclusión

Entender estos conceptos teóricos es esencial para aplicar eficazmente métodos basados en grids y hash multidimensional en problemas de acceso espacial. La teoría proporciona el fundamento necesario para diseñar, implementar y optimizar soluciones prácticas como el sistema de geolocalización de restaurantes descrito en el ejemplo.

ENFOQUE BASADO EN PROBLEMAS: SISTEMA DE GEOLOCALIZACIÓN PARA RESTAURANTES

Problema a Resolver

Necesitamos desarrollar un sistema que permita encontrar restaurantes cercanos a una ubicación dada en una ciudad. Utilizaremos un enfoque basado en grids para dividir el área geográfica en una cuadrícula y luego utilizaremos un hash multidimensional para gestionar y buscar eficientemente los restaurantes.

Paso 1: Representar los Datos

Primero, representaremos la ciudad como una cuadrícula y cada restaurante tendrá una posición geográfica (latitud y longitud). Asignaremos cada restaurante a una celda en la cuadrícula.

Definición de la Cuadrícula y los Restaurantes

```
class Restaurant {
constructor(name, latitude, longitude) {

this.name = name;
this.latitude = latitude;
this.longitude = longitude;
}
}

class Grid {
constructor(size) {
this.size = size; // tamaño de cada celda de la cuadrícula
```

```
this.cells = new Map();
}

// Función de hash para convertir coordenadas en una clave de celda
getCellKey(latitude, longitude) {
const x = Math.floor(latitude / this.size);
const y = Math.floor(longitude / this.size);
return `${x},${y}`;
}

// Añadir un restaurante a la cuadrícula
addRestaurant(restaurant) {
const key = this.getCellKey(restaurant.latitude, restaurant.longitude);
if (!this.cells.has(key)) {
this.cells.set(key, []);
}
this.cells.get(key).push(restaurant);
}

// Buscar restaurantes en una celda específica
getRestaurantsInCell(latitude, longitude) {
const key = this.getCellKey(latitude, longitude);
return this.cells.get(key) || [];
}

// Buscar restaurantes en celdas vecinas
getNearbyRestaurants(latitude, longitude) {
const x = Math.floor(latitude / this.size);
const y = Math.floor(longitude / this.size);
const nearbyRestaurants = [];
for (let dx = -1; dx <= 1; dx++) {
for (let dy = -1; dy <= 1; dy++) {
const key = `${x + dx},${y + dy}`;
if (this.cells.has(key)) {
nearbyRestaurants.push(...this.cells.get(key));
}
}
}
return nearbyRestaurants;
}
}
```

Paso 2: Poblar la Cuadrícula con Datos de Ejemplo

A continuación, agregaremos algunos restaurantes a la cuadrícula para poder realizar búsquedas.

```
// Crear una cuadrícula con tamaño de celda 0.01 (aproximadamente 1km x 1km)
const grid = new Grid(0.01);
```

```
// Añadir algunos restaurantes a la cuadrícula
const restaurants = [
    new Restaurant("Pizza Place", 40.712776, -74.005974),
    new Restaurant("Burger Joint", 40.713776, -74.005974),
    new Restaurant("Sushi Bar", 40.712776, -74.006974),
    new Restaurant("Pasta House", 40.713776, -74.006974),
    new Restaurant("Taco Stand", 40.714776, -74.005974),
    new Restaurant("Steak House", 40.714776, -74.006974)
];

restaurants.forEach(restaurant => grid.addRestaurant(restaurant));
```

Paso 3: Buscar Restaurantes Cercanos

Finalmente, implementaremos una función para buscar restaurantes cercanos a una ubicación específica y mostrar los resultados.

```
// Función para encontrar y mostrar restaurantes cercanos a una ubicación dada
function findNearbyRestaurants(latitude, longitude) {
    const nearbyRestaurants = grid.getNearbyRestaurants(latitude, longitude);
    console.log(`Restaurantes cercanos a (${latitude}, ${longitude}):`);
    nearbyRestaurants.forEach(restaurant => {
        console.log(`- ${restaurant.name} en (${restaurant.latitude},
${restaurant.longitude})`);
    });
}

// Buscar restaurantes cerca de una ubicación específica
const userLocation = { latitude: 40.713776, longitude: -74.005974 };
findNearbyRestaurants(userLocation.latitude, userLocation.longitude);
```

Explicación Pedagógica

Conceptos Clave

12. División Espacial con Grids:

- La ciudad se divide en una cuadrícula donde cada celda representa un área específica. Esto permite una organización eficiente de los datos espaciales.

13. Hash Multidimensional:

- Utilizamos una función de hash para mapear coordenadas geográficas a celdas en la cuadrícula. Esto simplifica la búsqueda de celdas y la inserción de datos.

14. **Búsqueda en Vecindad:**

    - Para encontrar restaurantes cercanos, no solo buscamos en la celda correspondiente a la ubicación del usuario, sino también en las celdas adyacentes. Esto mejora la precisión de la búsqueda en áreas de alta densidad.

Beneficios de Este Enfoque

15. **Eficiencia:**

    - La búsqueda en una estructura de cuadrícula es más rápida que una búsqueda lineal en todos los datos, especialmente cuando se trabaja con grandes conjuntos de datos.

16. **Escalabilidad:**

    - La estructura basada en cuadrículas es fácil de escalar y extender a medida que se agregan más datos.

17. **Simplicidad:**

    - El uso de una función de hash para mapear coordenadas a celdas es simple y directo, facilitando la implementación y el mantenimiento.

Este ejemplo práctico en JavaScript demuestra cómo los métodos basados en grids y hash multidimensional pueden ser aplicados para resolver problemas reales de acceso espacial de manera eficiente.