Tablas Hash en Python: Conceptos y Ejercicios

1. Conceptos Básicos de una Tabla Hash

Una tabla hash es una estructura de datos que asocia claves (keys) a valores (values) de manera eficiente. Se basa en una función hash que convierte las claves en índices en un arreglo subyacente, permitiendo el acceso rápido a los valores asociados a esas claves.

Funcionamiento Básico: La función hash toma una clave y devuelve un índice en el arreglo donde se almacenará el valor. Esta operación de conversión es la base de las tablas hash y es crucial para garantizar una búsqueda rápida, con un tiempo promedio de O(1).

Sin embargo, la eficiencia de las tablas hash depende en gran medida de dos factores:

1. **La calidad de la función hash**: Cuanto mejor distribuya la función hash las claves en el arreglo, mejor será el rendimiento de la tabla.
2. **El manejo de colisiones**: Las colisiones ocurren cuando dos claves diferentes generan el mismo índice. Un buen manejo de colisiones es esencial para mantener la eficiencia.

2. Importancia del Manejo de Colisiones

Cuando dos o más claves generan el mismo valor de índice (hash), se produce una colisión. Hay varias estrategias para manejar las colisiones, siendo las dos más comunes:

- **Encadenamiento (Chaining)**: En este enfoque, cada índice en la tabla contiene una lista enlazada de elementos que han generado el mismo índice. Si dos claves tienen el mismo hash, se agregan como elementos de la lista en ese índice.
- **Sondeo (Open Addressing)**: En este enfoque, cuando ocurre una colisión, se buscan otras posiciones en el arreglo para almacenar la clave, ya sea de manera secuencial (sondeo lineal) o de otra manera más compleja.

El manejo adecuado de colisiones es fundamental para evitar que las tablas hash se vuelvan ineficientes, especialmente cuando el número de colisiones aumenta. Si no se manejan correctamente, el tiempo de acceso a los elementos podría incrementarse, afectando negativamente la complejidad temporal.

3. La Importancia de una Buena Función Hash

Una función hash es responsable de convertir las claves en índices. Su calidad afecta directamente la distribución de los datos en la tabla y, por ende, el rendimiento de las operaciones de búsqueda, inserción y eliminación.

Características de una Buena Función Hash:

- **Distribución uniforme**: Una función hash eficiente distribuye las claves de manera uniforme a través del arreglo. Esto minimiza la posibilidad de colisiones y mejora el rendimiento general.
- **Determinística**: Para la misma clave, siempre debe devolver el mismo valor hash.
- **Rápida**: Debe calcularse rápidamente para no afectar el tiempo de las operaciones de la tabla hash.

Una función hash bien diseñada es crucial para evitar concentraciones de datos en algunas posiciones de la tabla y garantizar que las operaciones sean lo más eficientes posible.

Tipos de Funciones Hash

A continuación se describen 3 versiones de funciones hash que puedes implementar y usar para la tabla hash:

1. **Función Hash Multiplicativa (Método de Knuth)**:
    - **Descripción**: En este enfoque, se utiliza un número irracional llamado constante áurea para multiplicar el valor hash. Este enfoque se basa en el método de Knuth y busca distribuir mejor las claves.
    - **Algoritmo**: El valor hash de la clave se multiplica por la constante áurea inversa (aproximadamente 0.6180339887). Después de multiplicar, se toma la parte entera del resultado y se aplica el módulo con el tamaño de la tabla hash.
2. **Función Hash con Desplazamiento y XOR**:
    - **Descripción**: Esta función utiliza desplazamientos y la operación XOR para generar un valor de hash. Es una técnica que permite generar un hash con mejor dispersión y evitar agrupaciones en la tabla.
    - **Algoritmo**: Se toma un valor inicial para el hash (por ejemplo, 0) y para cada carácter de la clave, se aplica una combinación de desplazamientos a la izquierda y a la derecha junto con la operación XOR entre el valor actual del hash y el valor del carácter.
3. **Función Hash Personalizada (Propuesta del Ejercicio Original)**:
    - **Descripción**: Esta versión de la función hash convierte la clave en una cadena de bytes y suma los valores de estos bytes. Luego, el resultado se multiplica por un número primo (por ejemplo, 31) para mejorar la distribución, y finalmente se aplica el módulo con el tamaño de la tabla hash.
    - **Algoritmo**: Primero, la clave se convierte en una cadena de bytes. Luego, se suman los valores de estos bytes. El valor resultante se multiplica por 31 y se aplica el módulo con el tamaño de la tabla para obtener el índice final.

4. Ejercicio: Implementación de una Tabla Hash con Manejo de Colisiones

Objetivo: Implementar una tabla hash utilizando una función hash más compleja y manejar las colisiones usando encadenamiento.

Instrucciones:

1. Crea una clase `HashTable` que implemente la estructura de una tabla hash con los siguientes métodos:
    - `insert(key, value)`: Inserta un par clave-valor.
    - `search(key)`: Busca un valor asociado a la clave proporcionada.
    - `delete(key)`: Elimina un par clave-valor de la tabla.
    - `__str__()`: Devuelve una representación en forma de cadena de la tabla hash.
2. Implementa **3 versiones de la función hash** que se describen a continuación:

- **Función Hash Multiplicativa (Método de Knuth)**: Usa la constante áurea inversa para distribuir mejor las claves.
- **Función Hash con Desplazamiento y XOR**: Aplica desplazamientos y XOR para mejorar la dispersión.
- **Función Hash Personalizada**: Suma los valores de los bytes y aplica un número primo.

3. **Diseña tu propia función hash**: Los estudiantes deben diseñar su propia versión de una función hash, basada en lo aprendido, para generar una función eficiente que distribuya las claves de manera adecuada. Pueden basarse en las funciones proporcionadas, pero deben ser creativos en su diseño.

4. Utiliza **encadenamiento** para manejar las colisiones. Esto significa que, si dos claves tienen el mismo valor de hash, deben almacenarse en una lista enlazada dentro de esa posición de la tabla.

Requisitos adicionales: El tamaño de la tabla debe ser configurable (por defecto 10).

Prueba: Realiza pruebas insertando claves como `"a"`, `"b"`, `"c"`, luego realiza búsquedas y eliminaciones para verificar que todo funciona correctamente.

5. Ejercicio: Uso de la Tabla Hash

Objetivo: Utilizar la tabla hash para detectar duplicados en una lista y contar la frecuencia de los elementos.

Instrucciones:

1. Usa la clase `HashTable` implementada para resolver los siguientes problemas:

   a. **Detectar Duplicados:** Escribe una función `find_duplicates(arr)` que reciba una lista de elementos (números o cadenas) y devuelva los duplicados utilizando la tabla hash.

   b. **Contar Frecuencia:** Escribe una función `count_elements(arr)` que reciba una lista de elementos y devuelva un diccionario (o tabla hash) con la cantidad de veces que cada elemento aparece en la lista.

2. Realiza pruebas usando diferentes listas para verificar que la tabla hash se comporta correctamente en ambos casos.