### **Resolución de Colisiones en Tablas Hash**

Las **tablas hash** son estructuras de datos que permiten almacenar pares clave-valor de manera eficiente, logrando búsquedas, inserciones y eliminaciones en tiempo constante, en promedio. Sin embargo, para mantener esta eficiencia, las tablas hash deben resolver un problema frecuente: las **colisiones**.

Las colisiones ocurren cuando dos claves distintas son asignadas al mismo índice en el arreglo de la tabla hash debido a la función hash. A continuación, exploraremos las técnicas más comunes para manejar estas colisiones, sus ventajas, desventajas y algunos ejemplos.

#### **1. Métodos de Resolución de Colisiones**

Existen varias estrategias para resolver colisiones, cada una con sus propias aplicaciones según el contexto. A continuación, revisaremos las más utilizadas:

#### **1.1 Encadenamiento Separado (Separate Chaining)**

La técnica de encadenamiento, también conocida como **"separate chaining"**, consiste en almacenar múltiples elementos en una estructura de datos en cada índice de la tabla hash, generalmente usando una lista enlazada.

**Cómo funciona**: Cuando se produce una colisión, el nuevo elemento se agrega al final de la lista enlazada en ese índice. De esta forma, cada índice de la tabla actúa como el encabezado de una lista de elementos que comparten la misma posición de hash.

**Ventajas**:

* Sencillez en la implementación y flexibilidad en el almacenamiento de elementos.
* Facilita la expansión de la tabla hash, ya que los elementos solo deben reubicarse si cambian sus índices.

**Desventajas**:

* Puede consumir más memoria si la lista se llena de elementos.
* Cuando hay muchas colisiones, las listas pueden volverse largas, degradando el tiempo de búsqueda a O(n)O(n)O(n) en el peor de los casos.

**Ejemplo**: Si una tabla hash que almacena nombres recibe una colisión al agregar "Pedro" y "Pablo", ambos nombres se almacenan en una lista enlazada en el mismo índice. Cuando se busca "Pablo", se recorre esta lista hasta encontrar el valor correcto.

#### **1.2 Direccionamiento Abierto (Open Addressing)**

El direccionamiento abierto es un enfoque en el que las colisiones se resuelven al buscar una nueva posición dentro de la tabla misma en lugar de en una estructura auxiliar. Las estrategias de direccionamiento abierto más comunes son la **sondeo lineal** (linear probing), **sondeo cuadrático** (quadratic probing) y el **doble hash** (double hashing).

* **Sondeo Lineal**: En este método, cuando ocurre una colisión, se examinan las posiciones consecutivas en la tabla (incrementando de uno en uno) hasta encontrar una vacía.
* **Sondeo Cuadrático**: En lugar de avanzar de una en una, este método utiliza una secuencia cuadrática para definir el siguiente índice (por ejemplo, +1, +4, +9, etc.), ayudando a reducir el problema de clustering primario.
* **Doble Hashing**: Utiliza una segunda función hash para calcular un nuevo intervalo de sondeo cuando ocurre una colisión. Esto permite dispersar mejor los datos en la tabla, reduciendo la probabilidad de clustering.

**Ventajas**:

* Evita el uso de memoria adicional, ya que todos los elementos se mantienen dentro de la tabla.
* Puede ser eficiente en tablas con poca carga.

**Desventajas**:

* Si la tabla se llena o la carga es alta, puede volverse ineficiente.
* Requiere de un tamaño de tabla adecuado y una función hash que minimice colisiones para un mejor desempeño.

**Ejemplo**: Si al insertar "Pedro" ocurre una colisión en el índice 3, el método de sondeo lineal probará el índice 4, luego el 5, hasta encontrar una posición libre.

#### **1.3 Rehashing (Rehashing)**

Rehashing es una técnica en la que, ante una colisión, se aplica una segunda función hash para generar un nuevo índice para la clave. Este índice se calcula a partir de la clave, pero usando un método de hash diferente al original.

**Ventajas**:

* Es una técnica muy flexible y ayuda a reducir el problema de clustering.
* Puede manejar bien tablas con una carga más alta sin grandes impactos de desempeño.

**Desventajas**:

* La eficiencia depende de elegir adecuadamente las funciones de hash.
* Si las funciones no son bien diseñadas, pueden ocurrir patrones en los índices, aumentando las colisiones.

#### **2. Selección de la Estrategia de Resolución de Colisiones**

La elección del método adecuado para resolver colisiones depende de varios factores, como la carga esperada de la tabla hash, el tipo de datos y la necesidad de eficiencia en memoria o velocidad. A continuación, algunos criterios comunes:

* **Encadenamiento Separado**: Recomendado cuando la tabla hash necesita manejar una gran cantidad de elementos y se cuenta con memoria extra para listas adicionales. Es muy útil cuando se desconoce la cantidad final de elementos a almacenar.
* **Sondeo Lineal o Cuadrático**: Utilizados cuando la memoria es limitada y se puede gestionar la carga de la tabla hash (manteniendo, por ejemplo, una carga del 70% o menos).
* **Doble Hashing**: Apropiado cuando se busca minimizar el clustering, y es particularmente útil en situaciones en las que la tabla está casi llena.

#### **3. Ejemplo de Código para Encadenamiento Separado**

Aquí un ejemplo simple de una tabla hash con encadenamiento separado en Python:

class HashTable:

def \_\_init\_\_(self, size):

self.size = size

self.table = [[] for \_ in range(size)]

def hash\_function(self, key):

return hash(key) % self.size

def insert(self, key, value):

index = self.hash\_function(key)

for kvp in self.table[index]:

if kvp[0] == key:

kvp[1] = value # Actualiza si la clave ya existe

return

self.table[index].append([key, value])

def get(self, key):

index = self.hash\_function(key)

for kvp in self.table[index]:

if kvp[0] == key:

return kvp[1]

return None # No se encontró la clave

En este ejemplo, la función hash\_function calcula el índice y cada posición de la tabla contiene una lista para manejar colisiones mediante encadenamiento separado.

#### **4. Conclusión**

Las técnicas de resolución de colisiones son esenciales para mantener la eficiencia de las tablas hash. Aunque la función hash intenta distribuir las claves uniformemente, en la práctica, las colisiones son inevitables. La resolución de colisiones, ya sea por encadenamiento o direccionamiento abierto, garantiza que las tablas hash puedan gestionar grandes volúmenes de datos sin perder su eficiencia.

Esta introducción cubre las técnicas de resolución de colisiones y sus aplicaciones, lo que ayuda a ilustrar cómo abordar este problema clave en la gestión de datos en sistemas de alto rendimiento.