Tablas de dispersión - Hash Tables

Jorge Mario Londoño Peláez & Varias AI

May 5, 2025

1 Tablas de dispersión (Hash Tables)

Las tablas de dispersión, también conocidas como *hash tables* o tablas hash, son estructuras de datos que implementan el Abstract Data Type (ADT) tabla de símbolos (o diccionario) de manera eficiente. En esencia, una tabla hash permite almacenar y recuperar pares clave-valor de manera muy rápida. A diferencia de las implementaciones basadas en árboles de búsqueda, las tablas de dispersión no requieren que las llaves sean comparables, lo que las hace más versátiles. Son ampliamente utilizadas en diversas aplicaciones, desde bases de datos hasta compiladores, debido a su eficiencia en la búsqueda, inserción y eliminación de elementos.

La clave (key) utilizada en una tabla hash debe ser inmutable, es decir, su valor no debe cambiar después de ser insertada en la tabla. Esto es fundamental para garantizar que la función hash siempre produzca el mismo índice para la misma clave, permitiendo la correcta recuperación del valor asociado.

La idea central detrás de las tablas de dispersión es utilizar una *función hash* para mapear las llaves a los índices de un arreglo. Idealmente, esta función debería asignar cada llave a una posición única en el arreglo, permitiendo el acceso en tiempo casi constante. Sin embargo, en la práctica, es posible que diferentes llaves sean mapeadas a la misma posición, lo que se conoce como *colisión*. La gestión eficiente de estas colisiones es crucial para el rendimiento de la tabla de dispersión.

1.1 Funciones de dispersión (Hash Functions)

Una función de dispersión (o función hash) es una función que toma una llave como entrada y devuelve un valor entero, que se utiliza como índice en la tabla de dispersión. La elección de una buena función hash es crucial para el rendimiento de la tabla.

Propiedades deseables de una función de dispersión:

- **Uniformidad:** Debería distribuir las llaves de manera uniforme a lo largo de la tabla, minimizando las colisiones. Una función hash ideal asigna cada llave a una posición única en la tabla, pero esto rara vez es posible en la práctica.
- **Eficiencia:** Debería ser rápida de calcular, ya que se invoca en cada operación de inserción, búsqueda y eliminación. La eficiencia es especialmente importante en aplicaciones donde se realizan muchas operaciones en la tabla hash.
- **Determinismo:** Para una misma llave, la función hash siempre debe devolver el mismo valor. Esto es esencial para garantizar que la búsqueda de una llave siempre la encuentre en la misma posición.

Ejemplos de funciones de dispersión comunes:

- **Función de división (Division Method):** Esta función calcula el hash como el residuo de la división de la llave por el tamaño de la tabla: $h(k) = k \mod m$, donde k es la llave y m es el tamaño de la tabla. Es simple y rápida, pero puede generar muchas colisiones si el tamaño de la tabla es un múltiplo de un factor común de las llaves. Es recomendable que m sea un número primo.

- **Función de multiplicación (Multiplication Method):** Esta función calcula el hash multiplicando la llave por una constante A en el rango $0 < A < 1$, extrayendo la parte fraccionaria del resultado, y multiplicándola por el tamaño de la tabla: $h(k) = \lfloor m(kA \mod 1) \rfloor$, donde k es la llave, m es el tamaño de la tabla, y A es una constante. La elección de A es importante para la uniformidad de la distribución.

- **Funciones para cadenas de caracteres:** Para cadenas de caracteres, se pueden utilizar funciones que consideran cada carácter como un número y combinan estos números para generar el valor hash. Un ejemplo simple es sumar los valores ASCII de los caracteres, pero esto puede generar muchas colisiones. Funciones más avanzadas, como las que utilizan operaciones de desplazamiento de bits y XOR, pueden mejorar la distribución.

Ejemplo de función de dispersión para enteros (en Java):

```java
public int hash(int key, int tableSize) {
    return key % tableSize;
}
```

Listing 1: Función de dispersión simple para enteros

Ejemplo de función de dispersión no válida: Una función que siempre devuelve el mismo valor (e.g., `return 0;`) es una función de dispersión no válida, ya que causa que todas las llaves colisionen en la misma posición. Asimismo, una función de dispersión que para una llave de entrada no devuelva el mismo valor de salida, tampoco es válida. Por esta razón se exige que las funciones de dispersión sean consistentes con la igualdad: Llaves de igual valor siempre deben retornar el mismo hash.

Funciones de dispersión para otros tipos de datos: Las funciones de dispersión pueden diseñarse para diferentes tipos de datos, como cadenas de caracteres, objetos, etc. Para tipos de datos más complejos, es común combinar los valores hash de sus componentes. Cuando se implementa un ADT es importante sobre-escribir el método `hashCode` con el fin de implementar el cálculo de la función hash con base en el valor del ADT.

Funciones de dispersión vs. hash criptográficos: Es importante distinguir entre las funciones de dispersión utilizadas en las tablas de dispersión y las funciones hash criptográficas. Las funciones hash criptográficas están diseñadas para ser resistentes a colisiones y preimágenes, y se utilizan en aplicaciones de seguridad. Las funciones de dispersión para tablas de dispersión priorizan la eficiencia y la uniformidad, y no necesitan ser resistentes a ataques.

Algunos ejemplos de funciones hash criptográficas son: MD5, SHA.

1.2 Tablas de dispersión con encadenamiento (Hash Tables with Chaining)

El encadenamiento, también conocido como *separate chaining* (encadenamiento separado), es una técnica común para resolver colisiones en las tablas de dispersión. En esta técnica, cada posición del arreglo contiene una lista enlazada de todas las llaves que se mapean a esa posición. Es decir, en lugar de almacenar directamente el valor en la tabla, se almacena una lista de pares llave-valor que tienen el mismo índice hash.

Implementación de la tabla de dispersión con encadenamiento: La tabla de dispersión se implementa como un arreglo de listas enlazadas. Cuando se inserta una nueva llave, se calcula su valor hash, y se agrega la llave al principio de la lista enlazada en esa posición. La búsqueda y eliminación se realizan de manera similar, recorriendo la lista enlazada en la posición correspondiente.

Complejidad en el peor caso: En el peor caso, todas las llaves se mapean a la misma posición, y la tabla de dispersión se degenera en una lista enlazada. En este caso, las operaciones de inserción, búsqueda y eliminación tienen una complejidad de $O(n)$, donde n es el número de llaves en la tabla.

Factor de carga y redispersión: El factor de carga (load factor) de una tabla de dispersión se define como el número de llaves almacenadas dividido por el tamaño del arreglo. Un factor de carga alto indica que la tabla está "llena" y que las colisiones son más frecuentes. Para mantener un buen rendimiento, es común realizar una redispersión (rehashing) cuando el factor de carga excede un cierto umbral. La redispersión implica crear una nueva tabla de dispersión más grande, y reinsertar todas las llaves en la nueva tabla.

Complejidad con factor de carga acotado: Si el factor de carga se mantiene acotado (e.g., menor que 1), la complejidad promedio de las operaciones de inserción, búsqueda y eliminación es $O(1)$. Esto se debe a que la longitud promedio de las listas enlazadas es constante.

1.3 Direccionamiento abierto (Open Addressing)

Existen otras soluciones para el problema de las colisiones en las tablas de dispersión, además del encadenamiento. Una de las más comunes es el *direccionamiento abierto* (open addressing).

En esta técnica, todas las llaves se almacenan directamente en el arreglo. Cuando ocurre una colisión, se busca una posición vacía en el arreglo utilizando una función de *probing*. Algunas variantes comunes de direccionamiento abierto son:

- *Linear Probing:* Se busca la siguiente posición vacía en el arreglo de forma lineal. Si la posición está ocupada, se prueba la siguiente, y así sucesivamente, hasta encontrar una posición vacía.
- *Quadratic Probing:* Se busca una posición vacía utilizando una función cuadrática. Esto ayuda a evitar el agrupamiento primario que puede ocurrir con el linear probing.
- *Double Hashing:* Se utiliza una segunda función hash para determinar el intervalo entre las posiciones que se exploran. Esto proporciona una mejor distribución de las llaves que el *linear* y *quadratic probing*.

1.4 Cuckoo Hashing

Cuckoo Hashing es una técnica que utiliza dos funciones hash diferentes. Cuando se inserta una nueva llave, se calcula su posición utilizando la primera función hash. Si la posición está ocupada, la llave que estaba en esa posición se "desaloja" y se reinserta utilizando la segunda función hash. Este proceso se repite hasta que se encuentra una posición vacía o se alcanza un límite máximo de iteraciones. Si se alcanza el límite máximo de iteraciones, la tabla se redispersa.

1.5 Rendimiento de las tablas de dispersión

El rendimiento de una tabla de dispersión depende de varios factores, incluyendo la calidad de la función hash, la técnica de manejo de colisiones, y el factor de carga.

En general, las tablas de dispersión con encadenamiento y direccionamiento abierto tienen un rendimiento promedio de $O(1)$ para las operaciones de inserción, búsqueda y eliminación, siempre y cuando el factor de carga se mantenga acotado. Sin embargo, en el peor caso, el rendimiento puede ser de $O(n)$, donde n es el número de llaves en la tabla.

Cuckoo hashing puede ofrecer un rendimiento aún mejor en la práctica, pero su rendimiento en el peor caso es difícil de predecir.

1.6 Aplicaciones de las tablas de dispersión

Las tablas de dispersión tienen una amplia variedad de aplicaciones, incluyendo:

- **Bases de datos:** Las tablas de dispersión se utilizan para indexar las tablas de las bases de datos, permitiendo la búsqueda rápida de registros.
  B-Tree vs Hash index en MySQL
- **Compiladores:** Las tablas de dispersión se utilizan para implementar las tablas de símbolos de los compiladores, que almacenan información sobre las variables, funciones y otros elementos del programa.
- **Cachés:** Las tablas de dispersión se utilizan para implementar las cachés, que almacenan datos que se acceden con frecuencia para acelerar el acceso a los mismos.
  Ejemplos de servicios de caché implementados bajo esta filosofía son Redis y Memcached.
- **Diccionarios:** Las tablas de dispersión se utilizan para implementar los diccionarios, que son estructuras de datos que asocian claves con valores.