Santiago Reynaldo Aguilar Vega – A01709030

Jesús Ramírez Delgado - A01274723

Investigación y Reflexión: Códigos Hash

Una tabla hash o mapa hash es una estructura de datos que asocia llaves o claves con valores. La operación principal que soporta de manera eficiente es la búsqueda: permite el acceso a los elementos almacenados a partir de una clave generada usando el nombre, número de cuenta, id, etc.

Posibles implementaciones de los códigos Hash

Como el número de pares a manipular es indeterminado, se necesita una estructura de datos dinámica. Como primera solución podemos considerar una lista encadenada de pares. La operación de inserción es sencilla. Dado un número de pasaporte y una cadena, se crea un elemento nuevo en la tabla y se inserta. La operación de búsqueda necesita atravesar la lista hasta que, o se encuentra la clave dada (número de pasaporte), o se llega al final (la clave no está en la tabla). Cuando el número de pares es muy elevado, esta búsqueda es muy ineficiente, pues hay que procesar un gran número de elementos.

Una segunda opción para tener una búsqueda muy rápida podría ser almacenar todos los datos en una tabla y utilizar el número de pasaporte como su índice. En este caso, ver si un número está en la tabla consiste simplemente en ver si tiene una cadena asociada. El insertar un nuevo par (pasaporte, mensaje) también sería muy rápido pues consistiría en guardar el mensaje dado en la posición correspondiente al número. Pero el problema con esta solución es que el número de pasaportes puede ser un entero muy largo y por tanto la tabla que se necesita definir está más allá de lo aceptable.

La tabla hash ofrece un compromiso para esta situación. Los pares (clave, valor) se guardan en una tabla, pero con un tamaño menor del ideal. Para nuestro ejemplo, se utiliza una tabla, pero no tiene como tamaño el número máximo de pasaportes posibles, sino un número más pequeño.

Colisiones

Si dos llaves generan un hash apuntando al mismo índice, los registros correspondientes no pueden ser almacenados en la misma posición. En estos casos, cuando una casilla ya está ocupada, debemos encontrar otra ubicación donde almacenar el nuevo registro, y hacerlo de tal manera que podamos encontrarlo cuando se requiera.

Para dar una idea de la importancia de una buena estrategia de resolución de colisiones, considérese el siguiente resultado, derivado de la paradoja de las fechas de nacimiento. Aun cuando supongamos que el resultado de nuestra función hash genera índices aleatorios distribuidos uniformemente en todo el vector, e incluso para vectores de 1 millón de entradas, hay un 95% de posibilidades de que al menos una colisión ocurra antes de alcanzar los 2.500 registros.

Hay varias técnicas de resolución de colisiones, pero las más populares son encadenamiento y direccionamiento abierto.

Direccionamiento Cerrado, Encadenamiento separado o Hashing abierto: En la técnica más simple de encadenamiento, cada casilla en el array referencia una lista de los registros insertados que colisionan en la misma casilla. La inserción consiste en encontrar la casilla correcta y agregar al final de la lista correspondiente. El borrado consiste en buscar y quitar de la lista. La técnica de encadenamiento tiene ventajas sobre direccionamiento abierto. Primero el borrado es simple y segundo el crecimiento de la tabla puede ser pospuesto durante mucho más tiempo dado que el rendimiento disminuye mucho más lentamente incluso cuando todas las casillas ya están ocupadas. De hecho, muchas tablas hash encadenadas pueden no requerir crecimiento nunca, dado que la degradación de rendimiento es lineal en la medida que se va llenando la tabla. Por ejemplo, una tabla hash encadenada con dos veces el número de elementos recomendados, será dos veces más lenta en promedio que la misma tabla a su capacidad recomendada.

Las tablas hash encadenadas heredan las desventajas de las listas ligadas. Cuando se almacenan cantidades de información pequeñas, el gasto extra de las listas ligadas puede ser significativo. También los viajes a través de las listas tienen un rendimiento de caché muy pobre.

Otras estructuras de datos pueden ser utilizadas para el encadenamiento en lugar de las listas ligadas. Al usar árboles auto-balanceables, por ejemplo, el tiempo teórico del peor de los casos disminuye de O(n) a O(log n). Sin embargo, dado que se supone que cada lista debe ser pequeña, esta estrategia es normalmente ineficiente a menos que la tabla hash sea diseñada para correr a máxima capacidad o existan índices de colisión particularmente grandes. También se pueden utilizar vectores dinámicos para disminuir el espacio extra requerido y mejorar el rendimiento del caché cuando los registros son pequeños.

Direccionamiento abierto o Hashing cerrado: Las tablas hash de direccionamiento abierto pueden almacenar los registros directamente en el array. Las colisiones se resuelven mediante un sondeo del array, en el que se buscan diferentes localidades del array (secuencia de sondeo) hasta que el registro es encontrado o se llega a una casilla vacía, indicando que no existe esa llave en la tabla. Las secuencias de sondeo más socorridas incluyen:

Sondeo lineal: En el que el intervalo entre cada intento es constante (frecuentemente 1).
Sondeo cuadrático: En el que el intervalo entre los intentos aumenta linealmente (por lo que los índices son descritos por una función cuadrática), y
Doble hashe: En el que el intervalo entre intentos es constante para cada registro pero es calculado por otra función hash.

El sondeo lineal ofrece el mejor rendimiento del caché, pero es más sensible al aglomeramiento, en tanto que el doble hasheo tiene pobre rendimiento en el caché pero elimina el problema de aglomeramiento. El sondeo cuadrático se sitúa en medio. El doble hasheo también puede requerir más cálculos que las otras formas de sondeo.

Reflexión

Jesús:

Considero que esta actividad me ayudo a implementar bien los hashes y es una estructura de datos que en lo personal se puede llegar a parecer a un vector, en el sentido que puedes acceder fácilmente a los elementos de el mismo.

Santiago:

Los mapas o tablas hash son una estructura muy útil para poder acceder a sus elementos y esto se pudo corroborar mediante su implementación en este entregable 5.2. Creo que esta estructura de datos es muy buena y practica para mantener igual los datos ordenados y acceder a ellos fácilmente.

Referencias:

Geeks for Geeks. (2022). *Hashing Data Structure.* Geeks for Geeks. https://www.geeksforgeeks.org/hashing-data-structure/

Rodríguez N. (2022). *Métodos Principales de Hash Map.* Refactorizando. https://refactorizando.com/metodos-hashmap-java/