Tecnológico de Monterrey

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Programación de estructuras de datos y algoritmos fundamentales – TC1031

Act 5.2 - Actividad Integral sobre el uso de códigos hash (Evidencia Competencia)

Reflexión personal

Grupo 14

Adrián Alejandro Salgado Martínez – A00828843

28 de noviembre de 2020

Estructuras de datos por conjuntos

Las tablas de hash son una estructuras de datos bastante utilizada, cuyo principal propósito es almacenar y asociar un conjunto de datos a un conjunto de “llaves” con los que se pueden accesar, que debido al principio matemático y computacional que la fundamenta tiene propiedades que resultan ser tremendamente útiles para ciertas situaciones donde sus cualidades se aprovechen para reducir la complejidad computacional de un programa.

Entre los motivos por los cuales se suele usar una tabla de hash se encuentra que tanto el verificar si una llave existe en la tabla, como encontrar (para poder leerlo o modificarlo) el valor al que “mapea” esta llave, tiende a tener una complejidad constante. Esto se logra de manera tan eficiente gracias a la técnica que le da su nombre, conocida como “hashing”, donde para un espacio de memoria o un arreglo para asignar la ubicación de memoria donde residirá tanto la llave como el dato asociado se transforma matemáticamente la llave de tal manera que se distribuyan de la forma más uniforme posible, evitando las llamadas “colisiones” donde varias el hash de diferentes llaves lleva a la misma ubicación. A pesar de esto, el evitar por completo todo tipo de colisitones para todos los tipos de llaves es estadistica y prácticamente imposible, se deben de implementar técnicas para resolverlas, sea por dirección abierta, por encadenamiento, etc.

No obstante, algunas de las desventajas que presentan las hash tables son consecuencia de los mismos principios que les da sus atributos positivos, pues el hecho de utilizar una función de hashing no óptima para los tipos de datos con los que se trabaja, es decir, una que no distribuya de manera muy uniforme, significa que las colisiones aumentan considerablemente, y combinado a esto una técnica donde la resolución de colisiones no sea la más eficiente, como es el caso de la prueba lineal en la técnica de dirección abierta, significa que la complejidad de realizar las operaciones de pertenencia y búsqueda irán poco a poco a complejidad lineal. Asimismo, para minimizar la probabilidad usualmente es útil el uso de números primos, que introducen otro tipo de complejidad computacional para encontrarlos, y además de esto que son algo menos eficientes respecto al espacio pues para minimizar las colisiones deben existir espacios de memoria sin utilizar gracias al principio del palomar.

Concretamente, para esta solución problema se utilizó el `unordered_map` disponible en la librería estándar, el cual está implementado por medio de una tabla de hash. Sin embargo, a diferencia de los conjuntos que implementamos en clase donde solamente se guardaban los datos en un arreglo o en un vector, este está implementado por medio de “buckets”, o específicamente por medio de un arreglo de listas encadenadas de nodos, motivo por el cual es evidente que se puede decir que las colisiones son resueltas por encadenamiento. Entonces, al establecer nuestro `unordered_map` donde las llaves fueran strings y los valores fueran vectores, al utilizar el `unordered_map::operator[]` y `vector::push_back`, ambos con una complejidad `O(1)`, se logró tener una complejidad lineal general, y sobre todo una simplicidad en el código del programa honestamente sorprendente gracias a todo lo que tiene por ofrecer la librería estándar de C++.