Entregable 2: Tablas Hash

Estructuras de Datos 2024-1
Profesor José Fuentes Sepúlveda

Integrantes:
Diego Gutiérrez Mendoza — DIICC Ingeniería Civil Informática
Sofía López Aguilera — DIICC Ingeniería Civil Informática

1. Introducción

El objetivo de este entregable es comparar el rendimiento y tamaño que ocupan el unordered_map otorgado por la STL de C++ y las diferentes tablas hash que han sido implementadas de distintas maneras para el manejo de colisiones resultante de la función hash asociada cuando otorga el mismo resultado a distintas claves.

Las tablas tomaron como claves el User_ID y User_Name, cada clave en su tabla respectiva, y se implementaron con hashing abierto, cerrado con linear probing, cerrado con quadratic probing y cerrado con double hashing.

Como herramientas se utilizarán las librerías estándar de C + +, destacando “list” para las listas enlazadas del hashing abierto, “chrono” para hacer las comparaciones de tiempo y “unordered_map” para hacer la comparación con las funciones hash a implementar.

Se espera que las tablas con hashing abierto sean más rápidas para búsqueda y eliminación, sobre todo cuando la tabla se va llenando, mientras que las tablas con hashing cerrado sean más eficientes en espacio ya que utilizan solamente el arreglo para contener las llaves y los valores.

2. Estructuras de Datos Utilizadas

Tabla hash con hashing abierto: Maneja un arreglo que contiene las cabezas de listas enlazadas que contienen los valores de las claves que llegaron a esa posición con la función hash. Inician con 21997 buckets. La función hash de user_ID es aplicar módulo con key y size, y para user_name es sacar el valor acumulado de los carácteres en ASCII, multiplicarlo por 113 y aplicarle módulo size. Sus costes en el peor caso son O(n) para inserción y búsqueda.

Tabla hash con hashing cerrado y linear probing: Colocará el elemento en la siguiente celda disponible recorriendo el arreglo de forma circular. Inician con n*0.12 buckets, con n entregado al construirse. La función inicial hash de user_ID es aplicar módulo con key y size. Para la función inicial hash de user_name se utilizó la función de esta referencia "https://cp-algorithms.com/string/string-hashing.html". Luego para ambas funciones si hay colisión se le suma 1 al resultado y se aplica de nuevo el módulo, repitiendo el proceso hasta encontrar un espacio vacío. Sus costes en el peor caso son O(n) para inserción y búsqueda.

Tabla hash con hashing cerrado y quadratic probing: Al índice hash original se le añade valores sucesivos de un polinomio cuadrático arbitrario hasta que se encuentre un espacio vacío. Inician con n*0.12 buckets, con n entregado al construirse. Las funciones hash iniciales de user_ID y user_name se comparten entre las tres tablas de hashing cerrado. Si hay colisión, el índice original se le suma el cuadrado de i, que inicia en uno y va creciendo en uno también por cada colisión que encuentre, y se aplica módulo size. Sus costes en el peor caso son O(n) para inserción y búsqueda.

Tabla hash con hashing cerrado y double hashing: Al índice hash original se le aplica una segunda función hash para encontrar una posición vacía. Inician con n*0.12 buckets, con n entregado al construirse. Después de las funciones hashing iniciales, si hay colisión se aplica pero con otros valores la función de la referencia "https://cp-algorithms.com/string/string-hashing.html" para así obtener otra posición. Sus costes en el peor caso son O(n) para inserción y búsqueda.

Unordered_map: Es una implementación de una tabla hash de la librería estándar de C++, si el elemento encuentra una posición vacía se inserta, sino se almacena en una lista de colisiones. Sus costes en el peor caso son O(n) para inserción y búsqueda.

3. Dataset y Ambiente de Experimentación

Para leer el dataset se utilizó la librería “fstream” de C++, y se programó en Visual Studio Code.

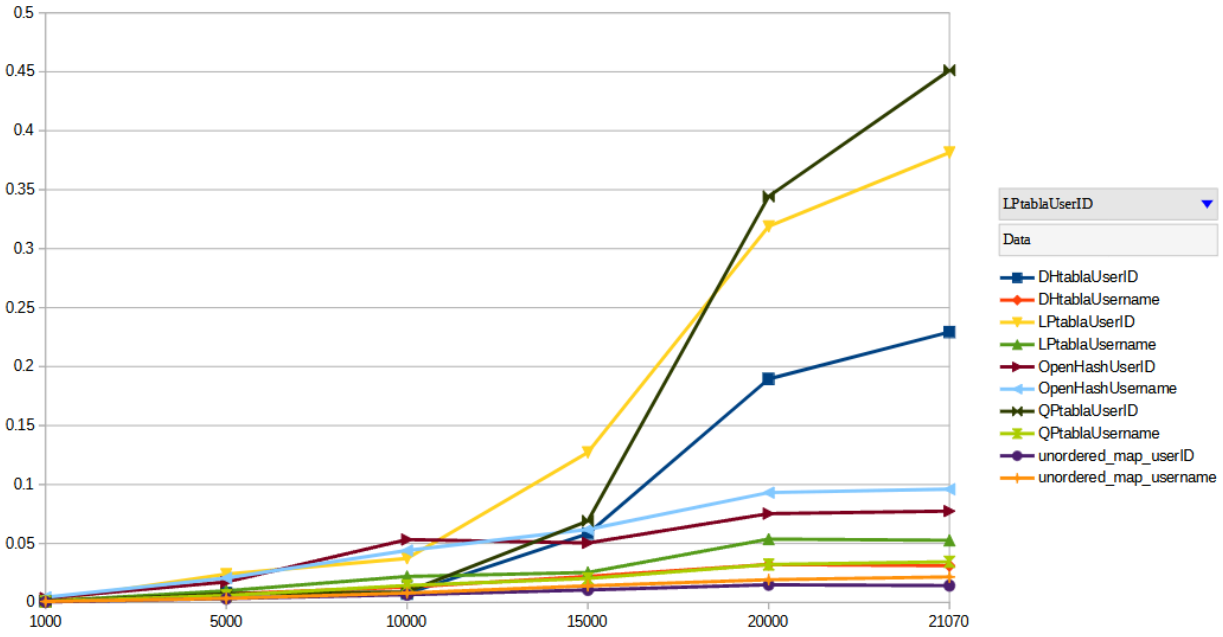
La máquina utilizada para la experimentación tiene un procesador AMD Ryzen 5 4600H con una frecuencia de 3 GHz, 20 GBs de Ram, Caché L1: 384 kB, Caché L2: 3,0 MB, Caché L3: 8,0 MB. Su sistema operativo es Windows 11 de 64 bits.

4. Resultados experimentales

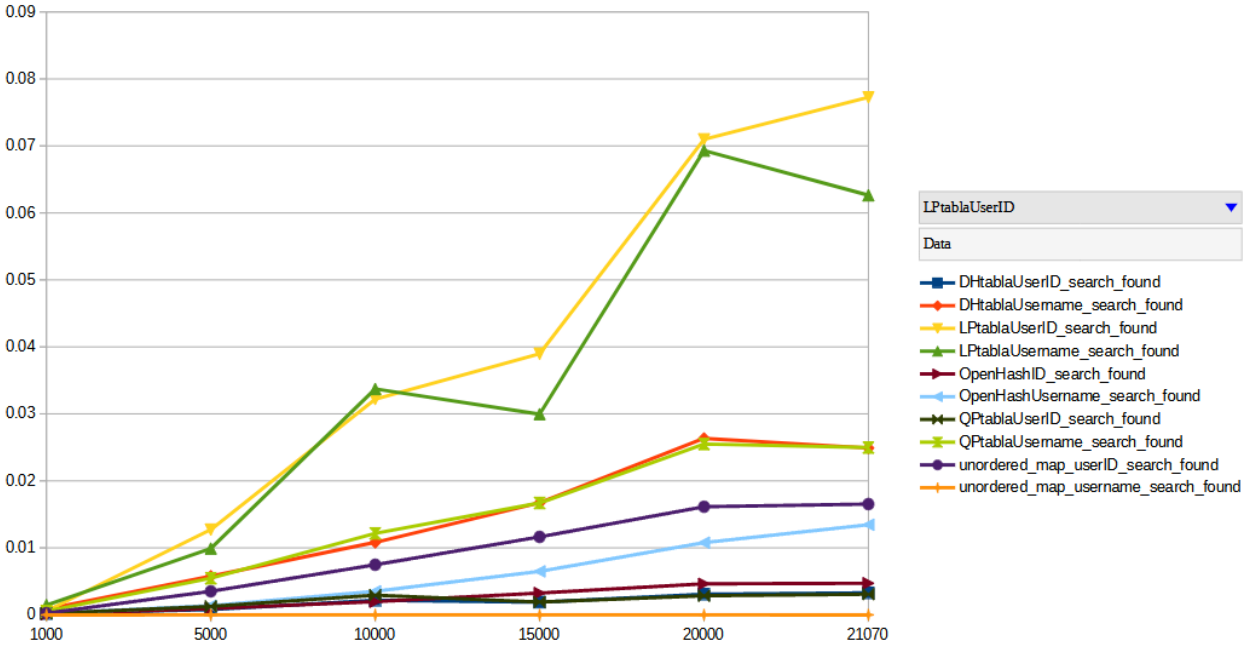
El tamaño para las tablas son las mismas tanto si toma como clave user_name y user_id. En la siguiente tabla se comparan cuanto espacio ocupan las tablas con un arreglo de tamaño 21997.

Estructura	Tamaño que ocupan en Bytes
HashOpen	2,375,676 Bytes
LP_HashCerrado	2,111,736 Bytes
QP_HashCerrado	2,111,736 Bytes
DH_HashCerrado	2,111,736 Bytes
Unordered_map	2,375,676 Bytes

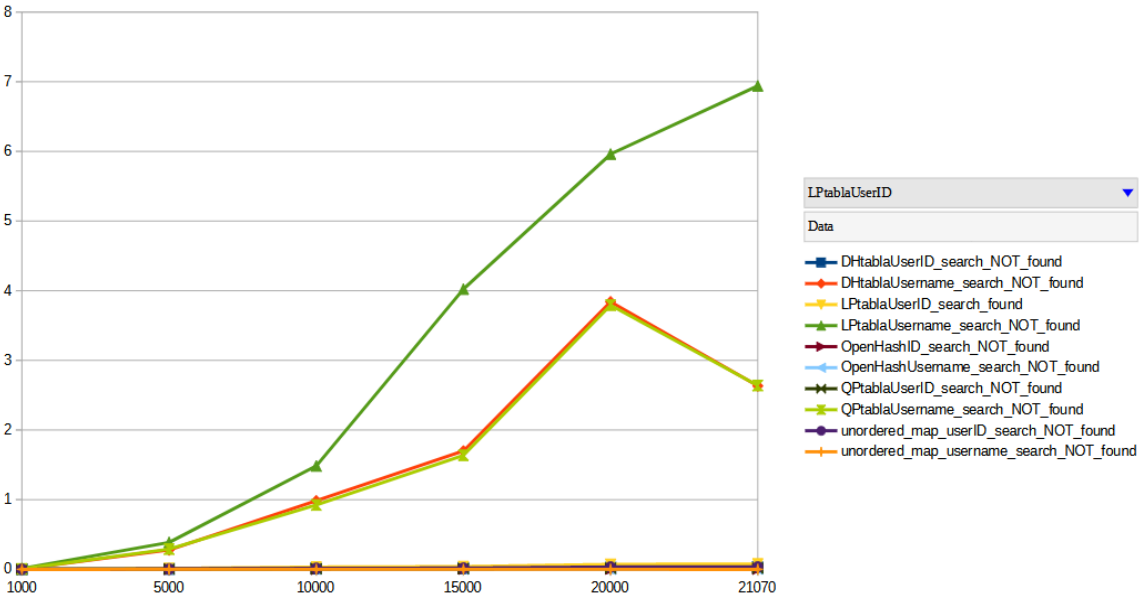
Insertar elementos tiempo promedio (segundos, cantidad de elementos)



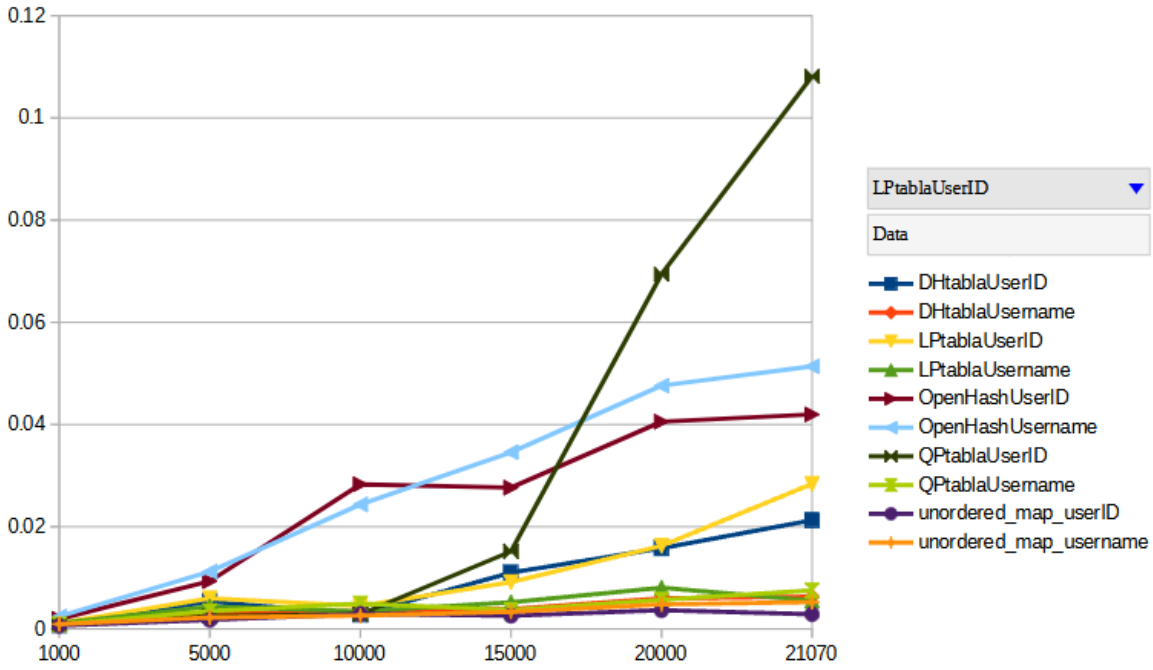
Búsqueda de elementos tiempo promedio (segundos, cantidad de elementos)



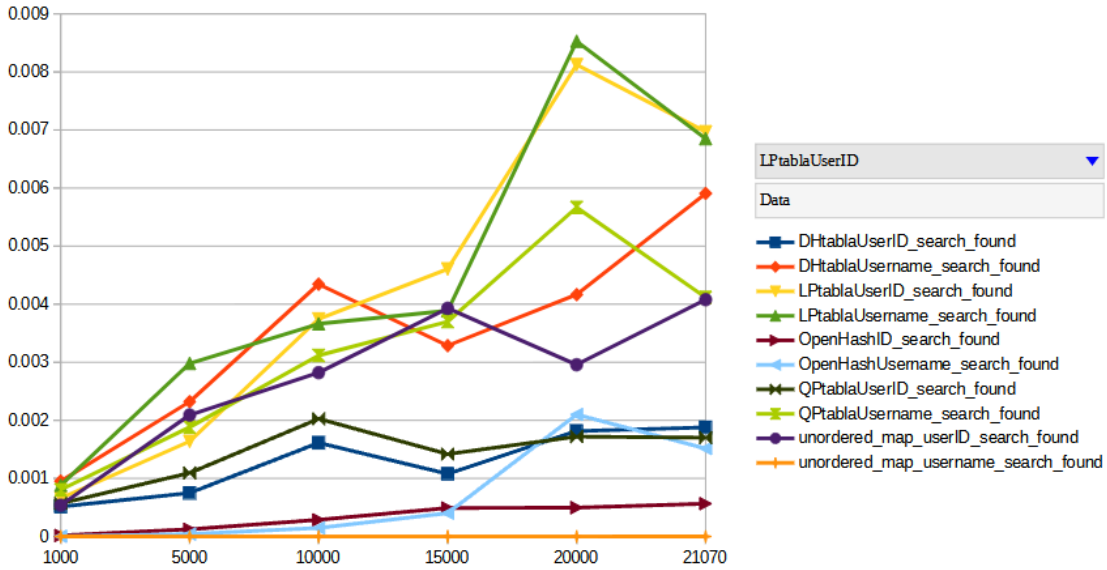
Búsqueda de elementos no encontrados tiempo promedio (segundos, cantidad de elementos)



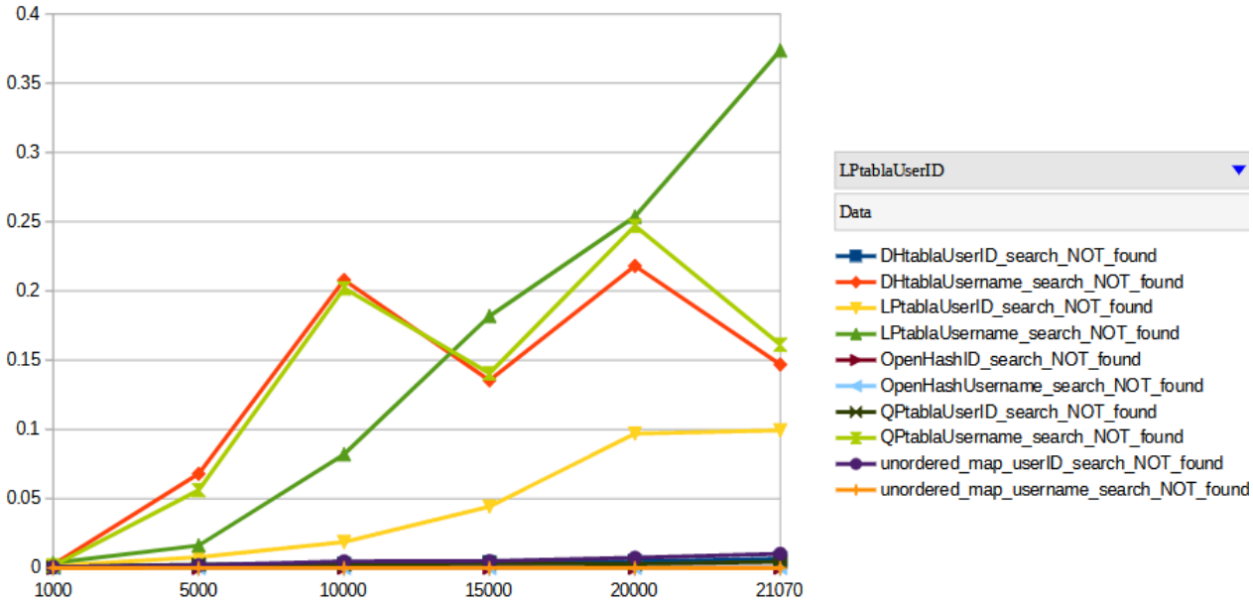
Insertar elementos desviación estándar (segundos, cantidad de elementos)



Búsqueda de elementos desviación estándar (segundos, cantidad de elementos)



Búsqueda de elementos no encontrados desviación estándar (segundos, cantidad de elementos)



5. Conclusiones

Se puede apreciar en los gráficos que para todos los métodos de hashing cerrado, tanto para Username y UserID como claves, el método cuadrático supera con creces al resto de los métodos una vez se ve más cercano al límite superior del tamaño de la tabla. Esto se debe a que el índice crece mucho más rápido que los otros dos métodos al encontrar una colisión, y al ser una muestra no muy grande de datos esto se ve reflejado en los resultados.

En cuanto a los métodos de hashing cerrado usando a Username como clave, se puede notar que es más rápido que el hashing con números enteros. Esto se explica por el método de función hash implementado en los strings, que es más refinado y minimiza colisiones al usar números primos grandes y generar un resultado más aleatorio de los datos.

Finalmente, como es de esperar, la implementación de Unordered_Map de la STL de C++ es más rápida que las implementaciones propias. En cuanto al tamaño, es lógico que sea igual al de HashOpen porque usan el mismo método de encadenamiento en listas ligadas para tratar colisiones.

Comparando el hashing abierto, el cerrado y el Unordered_map, se puede concluir que en cuanto al abierto es más eficiente en búsqueda e inserción pero utiliza más espacio. Se ve el caso contrario en el hashing cerrado. Unordered_map usa la misma memoria que el hashing abierto pero es más eficiente a nivel de velocidad de búsqueda e inserción.

6. Referencias

std::unordered_map - cppreference.com. (s. f.).

https://en.cppreference.com/w/cpp/container/unordered_map

Coding Jesus. (2019, 14 octubre). *C++ Hash Table Implementation* [Vídeo]. YouTube.

https://www.youtube.com/watch?v=2_3fR-k-LzI

String Hashing - Algorithms for Competitive Programming. (s. f.).

https://cp-algorithms.com/string/string-hashing.html

https://cplusplus.com/reference/fstream/fstream/
https://cplusplus.com/reference/unordered_map/unordered_map/