Preguntas del laboratorio

Grupo 1:

-María Camila Gómez Hernández – 202011050 – mc.gomezh1@uniandes.edu.co

-Nicolás Enrique Rueda Rincón -202013496 – ne.rueda@uniandes.edu.co

-Kevin Cohen Solano – 202011864 – k.cohen@uniandes.edu.co

Pregunta 1: Notan alguna diferencia en tiempo de carga y de consulta entre las dos implementaciones? ¿Si es así cuál es más rápida?

La verdad nuestro grupo no nota una gran diferencia numérica. Las dos implementaciones demoran en cargar alrededor de 47 segundos por lo que las dos son igual de efectivas. Algo que notamos diferente es que la creación del nuevo catálogo usando Separate Chaining demora mucho más que la creación de este mismo catalogo usando Linear Probing. Lo bueno de estas dos funciones es que el tiempo de las consultas en las dos es constante O(1).

Pregunta 2. Nota alguna diferencia en los tiempos de carga y/o de respuesta cuando el factor de carga cambia utilizando *Separate Chaining?* ¿Describa las diferencias encontradas?

Cuando aumentamos el factor de carga de Separate Chaining vemos que se demora mucho más el programa. Esto lo concluimos porque con un factor de carga de 0.5 demoraba 47 segundos y con un factor de 10 demoro 56.5 segundos. Esto se da porque por bucket están entrando más llaves-valores con lo que el tiempo de carga podría verse afectado, pues una buena distribución con factor de carga de 0.5 es O(1) pero una mala distribución hace que el factor de carga se acerque a O(n).

Pregunta 3. Nota alguna diferencia en los tiempos de carga y/o de respuesta cuando utiliza *Linear Probing* (factor de carga 0.5) y cuando el factor de carga de 10 en *Separate Chaining?* Describa las diferencias.

Cuando usamos un factor de carga de 10 en Separate Chaining, la velocidad de carga dura 56.5 segundos mientras que Linear Probing dura 47 segundos.

La principal diferencia de porque tarda más Separate Chaining es porque dentro de los buckets se esta guardando tanta información que termina habiendo una lista recargada de información, mientras Linear Probing aún mantiene su mecanismo de ir ordenando la información en una posición e intentar que haya espacios vacíos entre estos datos (“EMPTY”) para que los datos se repartan mejor, por lo que las búsquedas terminan siendo más fáciles O(1.5) que buscar en listas tremendamente grandes las cuales entre más grandes, más se acercan a O(n).