FEDA: Informe N3: Tablas hash vs arboles de búsqueda

Alumno: Diego Ignacio Neira Salgado

Profesores: José Fuentes y Cecilia Hernandez

Ayudantes: Víctor Cartes y Álvaro Guzmán

Introducción:

En el presente trabajo se tomo como base las estructuras siguientes estructuras tratadas en clases y los respectivos laboratorios:

- Árbol binario de búsqueda BST
- Tablas Hash cerradas
- Tablas Hash abiertas

Para cada una de estas estructuras se pidió escribir código que permita operar en sus realizando al menos las siguientes operaciones básicas:
Inserción eliminación y búsqueda (aunque la eliminación no se revisó experimentalmente también forma parte del código escrito y está disponible)

A la hora de diseñar las estructuras de las principales fuentes utilizadas fueron las diapositivas y apuntes. La escritura de código se basó en gran medida en los laboratorios del curso que se utilizaron como base y se modificaron para que se adaptaran al dataset.

Descripción de las estructuras:

Las estructuras se fueron en su forma principal idénticas a las vistas en los laboratorios 15, 17 y 18, por lo mismo en el caso de Hash Abierto se utilizo Murmur como base como familia de funciones.

Los detalles de las estructuras se pueden encontrar en los comentarios del código.

Descripción del dataset y el ambiente de experimentación.

Dataset

El dataset corresponde "TwitterFriends" de Kaggle un archivo CSV separado por comas cuyos campos fueron los que permitieron definir el struct “Usuarios” que se puede observar en Usuarios.h. Muestro a continuación un ejemplo de una línea (en particular la primera) del archivo:

id	1969527638
screenName	LlngoMakeEmCum_
tags	["#nationaldogday"]
avatar	http://pbs.twimg.com/profile_images/534286217882652672/FNmiQYVO_normal.jpeg
followersCount	319
friendsCount	112
lang	en
lastSeen	1472271687519
tweetId	769310701580083000
friends	["1969574754", "1969295556", "1969284056", "1969612214", "1970067476", "1969797386", "1969430539", "1969840064", "1969698176", "1970005154", "283011644", "1969901029", "1969563175", "1969302314", "1969978662", "1969457936", "1969667533", "1969547821", "1969943478", "1969668032", "283006529", "1969809440", "1969601096", "1969298856", "1969331652", "1969385498", "1969674368", "1969565263", "1970144676", "1969745390", "1969947438", "1969734134", "1969801326", "1969324008", "1969259820", "1969535827", "1970072989", "1969771688", "1969437804", "1969507394", "1969509972", "1969751588", "283012808", "1969302888", "1970224440", "1969603532", "283011244", "1969501046", "1969887518", "1970153138", "1970267527", "1969941955", "1969421654", "1970013110", "1969544905", "1969839590", "1969876500", "1969674625", "1969337952", "1970046536", "1970090934", "1969419133", "1969517215", "1969787869", "1969298065", "1970149771", "1969422638", "1969504268", "1970025554", "1969776001", "1970138611", "1969316186", "1969547558", "1969689272", "283009727", "283015491", "1969526874", "1969662210", "1969536164", "1969320008", "1969893793", "1970158393", "1969365936", "1970194418", "1969942094", "1969631580", "1969704756", "1969920092", "1969712882", "1969791680", "1969408164", "1969754851", "1970205480", "1969840267", "1969443211", "1969706762", "1969692698", "1969751576", "1969486796", "1969286630", "1969686674", "1969833492", "1969294814", "1969472719", "1969685018", "283008559", "283011243", "1969680078", "1969545697", "1969646412", "1969442725", "1969692529"]

Una cosa particular que no se aprecia y que generó bastantes problemas a la hora de la lectura fue que cada línea, así como varios elementos, comenzaban y terminaban con un signo “. Elemento que se debió eliminar en el código para poder, por ejemplo convertir los ID en números y poder operar sobre ellos así.

Esta necesidad de verificación generó la necesidad de ir probando si se podía procesar genero la necesidad de operar y presentó la idea (u oportunidad) de

almacenar los datos en un vector de usuarios que a partir de este punto será considerada la fuente de los datos (reemplazando como fuente al archivo .csv a la hora de trabajar en las estructuras)

Ambiente

El código fue escrito y ejecutado en un laptop de con procesador ARM64 (Snapdragon) con 32GB de memoria RAM corriendo WSL2 sobre Windows 11 Pro. Tomando como precaución cerrar todos los programas adicionales para reducir el ruido.

Resultados experimentales

Tamaño de las estructuras de datos, expresadas en B:

TAMAÑOS EXPERIMENTALES							
		Hcerr_ID	Hcerr_SN	Habier_ID	Habier_SN	BST_ID	BST_SN
1000	9461429	26672	26672	40008	40008	24000	24000
5000	60762322	133344	133344	200016	200016	120000	120000
10000	115794020	266672	266672	400008	400008	240000	240000
15000	171133622	400016	400016	600024	600024	360000	360000
20000	225267886	533344	533344	800016	800016	480000	480000
30000	337987014	800016	800016	1200024	1200024	720000	720000

Rendimiendo de las estructuras en creación/inserción:

TIEMPOS DE INSERCIÓN							
		Hcerr_ID	Hcerr_SN	Habier_ID	Habier_SN	BST_ID	BST_SN
1000	med	46,34	35,70	106,22	64,04	63,14	73,60
	var	13,41	4,26	37867,89	1836,20	1223,92	1126,57
5000	med	228,76	185,60	269,72	179,18	417,86	513,00
	var	221,37	37,10	2977,31	71,66	240,82	3364,65
10000	med	455,32	379,00	522,04	366,60	1059,12	1354,22
	var	523,53	128,20	8411,79	4780,08	14997,09	45478,75
15000	med	702,90	569,76	822,78	552,54	2139,52	2766,34
	var	1334,50	1879,49	17964,18	3902,09	371506,54	1090517,49
20000	med	908,44	764,80	1139,10	849,18	2609,52	3278,12
	var	778,01	724,45	59209,89	56810,27	117444,09	420679,94
30000	med	1380,44	1139,22	1878,32	1910,84	5432,92	7040,08
	var	12343,39	1157,28	10141,45	49189,65	189888,93	1282642,93

Rendimiento de las estructuras en búsqueda:

	MEDIA	VARIANCIA
Hash Cerrado busqueda exitosa por ID	7,96	57,35
Hash Cerrado busqueda exitosa por ScreenName	4,02	7,69
Hash Abierto busqueda exitosa por ID	7,92	53,99
Hash Abierto busqueda exitosa por ScreenName	5,3	26,62
BST busqueda exitosa por ID	17,64	332,68
BST busqueda exitosa por ScreenName	31,58	716,78
Hash Cerrado busqueda fallida por ID	6,24	19,29
Hash Cerrado busqueda fallida por ScreenName	11,9	80,74
Hash Abierto busqueda fallida por ID	4,3	7,40
Hash Abierto busqueda fallida por ScreenName	8,46	19,23
BST busqueda fallida por ID	0,32	0,22
BST busqueda fallida por ScreenName	9,04	4,94

Conclusiones

De acuerdo a los resultados experimentales se puede observar que en este set de datos que Hash Cerrado obtiene los mejores resultados cuando los datos que se buscan si están en el dataset, en particular se observa un muy buen desempeño cuando se utiliza el ScreenName, probablemente porque al tratarse de strings que son mas complejos, es más fácil que se rompan patrones y se generen menos colisiones. El BST parece tomar mas tiempo cuando el elemento si se encuentra en el dataset, lo que era esperable ya que debe realizar el recorrido del árbol para encontrarlos.

Llama la atención que cuando los elementos no se encuentran logra buenos tiempos. Se puede especular que esto se debe a la forma en que se generaron de los elementos “no existentes” a buscar, que correspondían a números pequeños (i) y a variaciones y a una adición de un string sencillo (que incluía una @ que deduje sería un carácter prohibido en la creación de ScreenName en la plataforma pero no genera conflictos en el código), lo que probablemente hizo que se llegara rápidamente a una hoja del árbol, probablemente un AVL podría empeorar esos resultados (de los no existentes) pero daría mejores resultados y mas estables pero en los datos si existentes.

Oportunidades de mejora:

Se podría mejorar para aplicaciones que no se solicitaban la forma en que se generó el vector de Usuarios separar en cada uno de sus elementos los campos de hashtags y de amigos, cosa que no se realizó en el presente trabajo en virtud del tiempo y ya que no se utilizarían ni entregan ninguna ventaja para los objetivos de la tarea.

También se debería encontrar otra forma mas aleatoria de generar el set de elementos no presentes para que representen de mejor manera el rendimiento de las estructuras, aunque estos resultados también son significativos y entregan información adicional.