Análisis práctico de las operaciones de inserción, búsqueda y eliminación en estructuras de datos

Jean Aguilar Mena, B70134, jean.aguilarmena@ucr.ac.cr

***Resumen*—En este trabajo se aborda el estudio y comparación de distintas estructuras de datos, como listas simplemente enlazadas, listas doblemente enlazadas, árboles de búsqueda binaria, árboles rojinegros, tablas de dispersión (encadenamiento) y tablas de dispersión (direccionamiento abierto), o comúnmente conocidas como tablas hash. Para esto, se generaron elementos aleatorios y elementos ordenados, con el fin de medir las operaciones básicas y elementales de estas, como lo son la inserción, búsqueda y eliminación de elementos. Para ellos se insertaron en cada estructura de datos un total de 1 millón de elementos, este proceso se ejecutó 3 veces para cada tipo de inserción, aleatoria y ordenada, para lo cuál, se midió los tiempos de ejecución de cada operación en 3 corridas, calculando ele promedio para cada caso. El resultado fue que las estructuras de datos lineales, tardaron bastante más realizando las operaciones que las estructuras mejor balanceadas y que la dierenciación entre inserción aleatoria u ordenada, se refleja más en algunas estructuras que en otras, y que en general suelen influir más en el tiempo de inserción que en el tiempo de búsqueda o eliminación.**

***Palabras clave*—lista, árboles binarios, tablas hash, inserción, búsqueda, eliminación.**

I. Introducción

El propósito del trabajo es comparar empíricamente el rendimiento de varias estructuras de datos cuando se realizan operaciones básicas (inserción, búsqueda y eliminación), y examinar cómo cambia dicho rendimiento cuando los elementos se insertan en orden aleatorio frente a un orden definido (ascendente o descendentemente). Es importante aclarar que este trabajo no pretende demostrar resultados teóricos ni establecer nuevos teoremas, su objetivo es observar y documentar el comportamiento práctico de las implementaciones con conjuntos de datos representativos y discutir, en qué medida concuerdan los resultados obtenidos, con la teoría conocida y estudiada, haciendo referencia al caso particular que se está trabajando, destacando que otros análisis parecidos, podrían concluir resultados diferentes, por n cantidad de factores.

Además es importante destacar que si bien es cierto, las estructuras de datos son implementaciones propias, basadas en el entendimiento de estas, también, se toma mucho como referencia la teoría del libro "Introduction to algorithms" de Thomas Cormen...

Se estudian las siguientes estructuras:

- Lista simplemente enlazada (SLList).
- Árbol de búsqueda binaria (BSTree).
- Árbol rojinegro (RBTree).
- Tabla de dispersión con encadenamiento (ChainedHashTable), con una lista doblemente enlazada como estructura de encadenamiento.
- Tabla de dispersión con direccionamiento abierto (OpenAddressHashTable), con sondeo de doble hash.

Ante esto se propone abarcar 2 objetivos claros:

Objetivos

1. **Verificar la coherencia entre resultados experimentales y la teoría:** Evaluar si los tiempos observados para inserción, búsqueda y eliminación en cada estructura (lista enlazada, árbol de búsqueda binaria, árbol rojinegro, tablas de dispersión encadenadas y direccionamiento abierto) se alinean con el rendimiento esperado en los escenarios de elementos aleatorios y ordenados.
2. **Comparar el comportamiento práctico entre estructuras en búsquedas y eliminaciones:** Identificar cuáles estructuras presentan diferencias significativas entre inserción ordenada y aleatoria y que tanto responden a lo esperado por teoría.

II. Metodología

La lista simplemente enlazada se utilizó con inserción al inicio para asegurar un tiempo constante en esta operación, no obstante, con el fin de evitar elementos duplicados, se incorporó una búsqueda previa antes de insertar, lo que provoca que el tiempo total dependa directamente del tamaño de la lista (O(n)). Para el manejo de árboles, se empleó un Árbol de Búsqueda Binaria (ABB) que incorpora una opción de fast insert, mecanismo orientado a mantener el árbol balanceado al insertar datos que podrían provocar el peor caso de un árbol de búsqueda binaria convencional (como secuencias ordenadas).

En cuanto al manejo de colisiones mediante dispersión, se trabajó con dos enfoques. La tabla de dispersión con encadenamiento se construyó utilizando un vector de listas doblemente enlazadas con nodo centinela, incrementando el contador de colisiones (collisionCount) cuando el bucket de inserción no se encontraba vacío. Durante los procesos de búsqueda y eliminación, dicho contador también aumentó cuando el bucket contenía más de un elemento, y con el propósito de obtener mediciones independientes, fue reseteado entre las fases de inserción, búsqueda y eliminación. La tabla de dispersión con direccionamiento abierto se implementó mediante un vector de valores y un vector paralelo de estados para identificar las celdas como empty, occupied o deleted. Esta estructura empleó una función hash principal basada en middle-squared combinada con una secuencia de Weyl para la dispersión inicial, y una segunda función hash definida como h2(k) = 1 + (k

Cuadro I
TIEMPOS DE INSERCIÓN.

		Tiempo (Milisegundos)			
		Corrida			
Algoritmo	Tam. (k)	1	2	3	Prom.
LSD	Ordenada	34	21	30	28
	No ordenada	554241	548585	540964	547930
ABB	Ordenada	46	39	37	41
	No ordenada	290	287	277	285
ARN	Ordenada	163	148	146	152
	No ordenada	379	552	370	434
TDE	Ordenada	31	28	27	29
	No ordenada	141	144	139	141
TDDA	Ordenada	7	8	8	7,67
	No ordenada	49	47	46	47,33

Cuadro II
TIEMPOS DE BÚSQUEDA.

		Tiempo (Milisegundos)			
		Corrida			
Algoritmo	Tam. (k)	1	2	3	Prom.
LSD	Ordenada	10420	10279	10295	10331
	No ordenada	10137	9925	9904	9989
ABB	Ordenada	1	1	1	1
	No ordenada	3	3	3	3
ARN	Ordenada	3	2	2	2
	No ordenada	2	2	2	2
TDE	Ordenada	3	1	1	2
	No ordenada	0	0	0	0
TDDA	Ordenada	0	1	0	0,33
	No ordenada	0	0	0	0

Cuadro III
TIEMPOS DE ELIMINACIÓN.

		Tiempo (Milisegundos)			
		Corrida			
Algoritmo	Tam. (k)	1	2	3	Prom.
LSD	Ordenada	10522	10408	10353	10428
	No ordenada	10318	10063	10048	10143
ABB	Ordenada	1	1	1	1
	No ordenada	6	6	7	6
ARN	Ordenada	2	2	1	2
	No ordenada	2	2	2	2
TDE	Ordenada	1	0	0	0
	No ordenada	1	1	1	1
TDDA	Ordenada	1	1	0	0,67
	No ordenada	0	0	1	0,33

Los valores insertados corresponden a enteros con signo de 32 bits. Para la generación de los datos se consideraron dos estrategias: Una basada en números aleatorios y otra basada en valores secuenciales. La ejecución de las operaciones de inserción, búsqueda y eliminación sobre cada estructura fue gestionada a través de una clase auxiliar encargada de ejecutar dichas acciones en orden, garantizando la independencia de los datos utilizados en cada estructura. Los valores almacenados variaron entre estructuras y entre corridas, ya que cada una utilizó sus propios generadores y no se cargaron datos desde archivos externos, además, cada estructura fue evaluada tres veces por operación. Cabe destacar que inicialmente se probó el uso de sondeo cuadrático en la tabla con direccionamiento abierto, pero debido a la alta cantidad de colisiones obtenidas, se sustituyó por double hashing, lo que permitió estabilizar significativamente dichos valores.

III. RESULTADOS

El cuadro 1 tiene los resultados de la operación de inserción de las 3 corridas de cada una de las estructuras de datos implementadas, mostrando de forma numérica la diferencia marcada entre una estructura y otra, además de la diferencia entre inserción ordenada y no ordenada. Los tiempos están en unidades de milisegundos, unidad lo suficiente pequeña para evidenciar la diferencia entre uno y otro, pero lo suficientemente grande para no tener datos extremadamente grandes. Es importante, destacar que estos resultados provienen de ejecutar cada algoritmo 3 veces, cada ejecución con inserciones de valores diferentes, pero también diferentes para todas las estructuras, por lo que es posible que haya alguna diferencia por esa parte también.

El cuadro 2 tiene los resultados de la operación de búsqueda de las 3 corridas de cada una de las estructuras de datos implementadas, mostrando de forma numérica la diferencia marcada entre una estructura y otra, además de la diferencia entre inserción ordenada y no ordenada. Los tiempos están en unidades de milisegundos, unidad lo suficiente pequeña para evidenciar la diferencia entre uno y otro, pero lo suficientemente grande para no tener datos extremadamente grandes. Es importante, destacar que estos resultados provienen de ejecutar cada algoritmo 3 veces, cada ejecución con inserciones de valores diferentes, pero también diferentes para todas las estructuras, por lo que es posible que haya alguna diferencia por esa parte también, que se vea reflejadaa en las diferencias, aunque al ejecutarlo 3 veces, se minimiza esa posibilidad.

El cuadro 3 tiene los resultados de la operación de eliminación de las 3 corridas de cada una de las estructuras de datos implementadas, mostrando de forma numérica la diferencia marcada entre una estructura y otra, además de la diferencia entre inserción ordenada y no ordenada. Los tiempos están en unidades de milisegundos, unidad lo suficiente pequeña para evidenciar la diferencia entre uno y otro, pero lo suficientemente grande para no tener datos extremadamente grandes. Es importante, destacar que estos resultados provienen de ejecutar cada algoritmo 3 veces, cada ejecución con inserciones de valores diferentes, pero también diferentes para todas las estructuras, por lo que es posible que haya alguna diferencia por esa parte también, que se vea reflejadaa en las diferencias, aunque al ejecutarlo 3 veces, se minimiza esa posibilidad.

Es importante aclarar que los rangos de los gráficos a continuación, están en muchos casos en una escala logarítmica, esto para poder tener una representción en un rango más adecuado para visualizar las diferencias entre una estructura y otra.

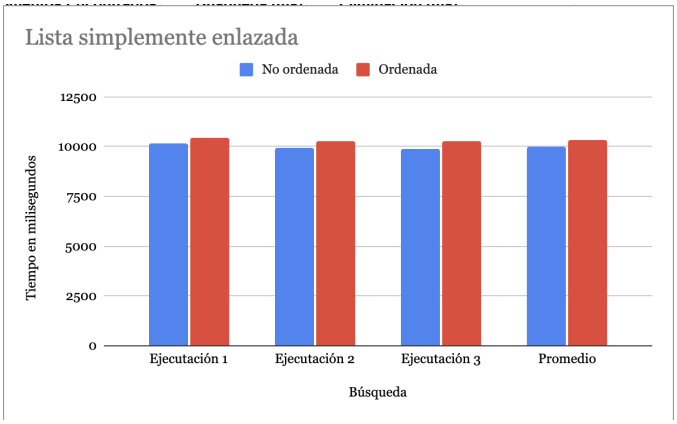


Figura 1. Tiempos de búsqueda en lista simplemente enlazada para inserción ordenada y no ordenada.

En la figura 1, se observa que los tiempos de búsqueda en una lista simplemente enlazada para inserción ordenada e inserción no ordenada (sin repetidos) es muy parecida y se acerca para cada una de las ejecuciones a los 10,000 milisegundos, para una búsqueda de 10,000 elementos.

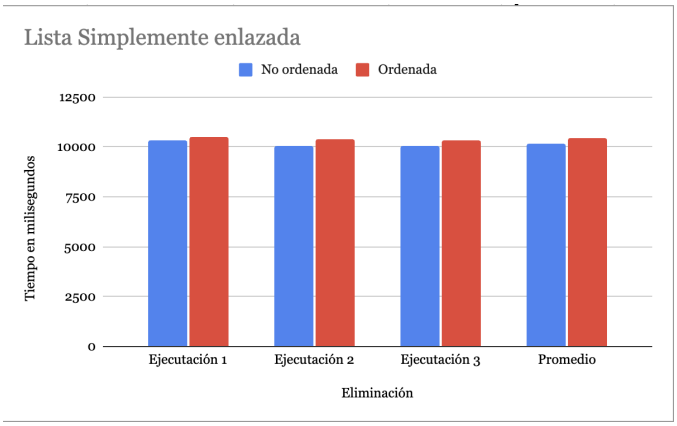


Figura 2. Tiempos de eliminación en lista simplemente enlazada para inserción ordenada y no ordenada.

En la figura 2, se observa que los tiempos de eliminación en una lista simplemente enlazada, al igual que para búsqueda, son muy parecidos entre una lista ordenada y una lista no ordenada, rondando igual un promedio de 10,000 milisegundos, para una eliminación de 10,000 elementos.

En la figura 3, se puede observar una diferenciación clara entre búsqueda para elementos insertados en orden (balanceado con fast insert) y no ordenado. La búsqueda de 10,000 elementos a partir de una inserción ordenada tardó para cada una de las ejecuciones, 1 milisegundo, mientras que para elementos insertados de manera no ordenada, tardó 3 milisegundos.

En la figura 4, se observa una clara diferenciación entre los tiempos de eliminación para elementos insertados de manera

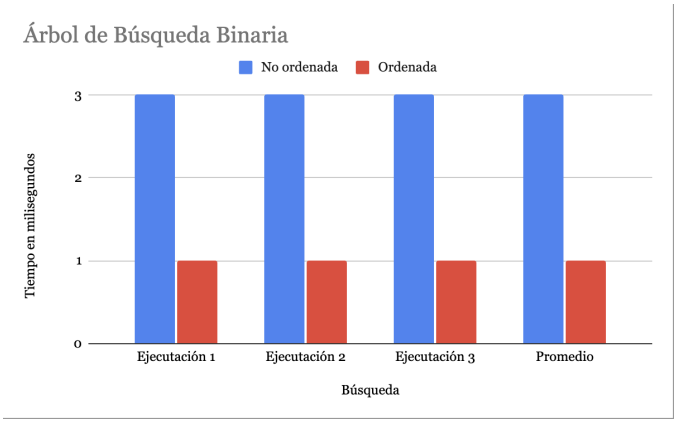


Figura 3. Tiempos de búsqueda en un árbol de búsqueda binaria con inserción ordenada y no ordenada.

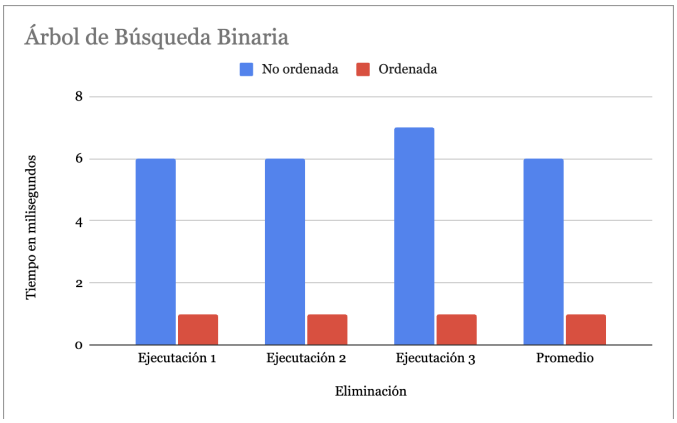


Figura 4. Tiempos de eliminación en un árbol de búsqueda binaria con inserción ordenada y no ordenada.

ordenada en comparación con elementos insertados de manera no ordenada. Para el caso de eliminación de 10,000 elementos insertados de manera ordenada (balanceado con fast insert), los tiempos de todas las ejecuciones alcanzan apenas los 0,33 milisegundos, mientras que para la eliminación de los elementos insertados de manera no ordenada, en promedio tardó 6 milisegundos.

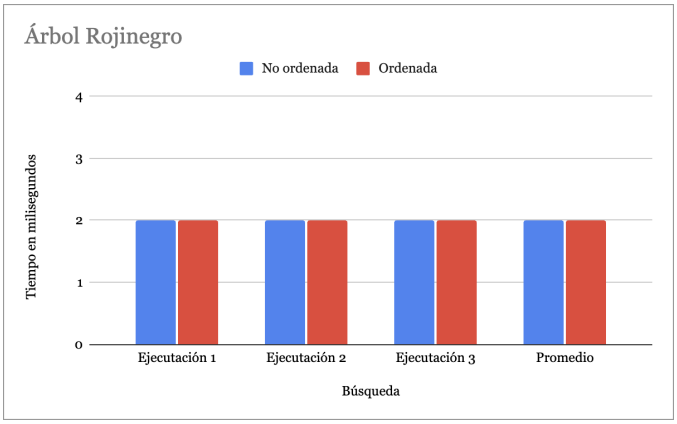


Figura 5. Tiempos de búsqueda en un árbol rojo-negro con inserción ordenada y no ordenada.

En la figura 5, se presenta los tiempos de búsqueda de 10,000 elementos, para elementos insertados de manera ordenada y de manera no ordenada. Para ambos tipos de inserción, el promedio de búsqueda de 10,000 elementos, es exactamente 2 milisegundos.

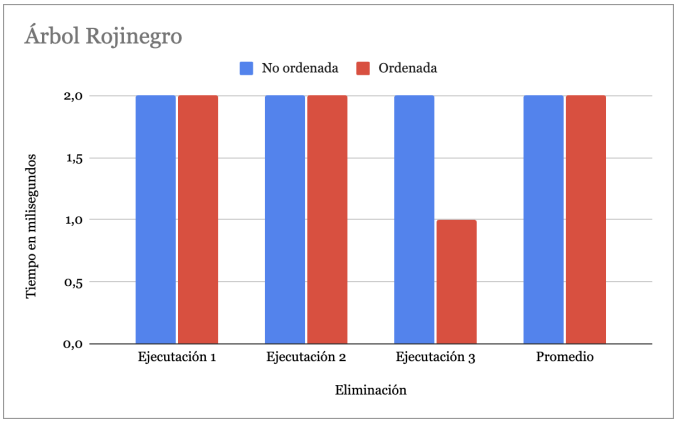


Figura 6. Tiempos de eliminación en un árbol rojo-negro con inserción ordenada y no ordenada.

En la figura 6, se presenta los tiempos de eliminación de 10,000 elementos, para elementos insertados de manera ordenada y de manera no ordenada. Para ambos, el promedio ronda los 2 milisegundos, con la excepción de la ejecución 3, donde el tiempo de eliminación de 10,000 elementos a partir de einserción ordenada, tardó 1 milisegundo.

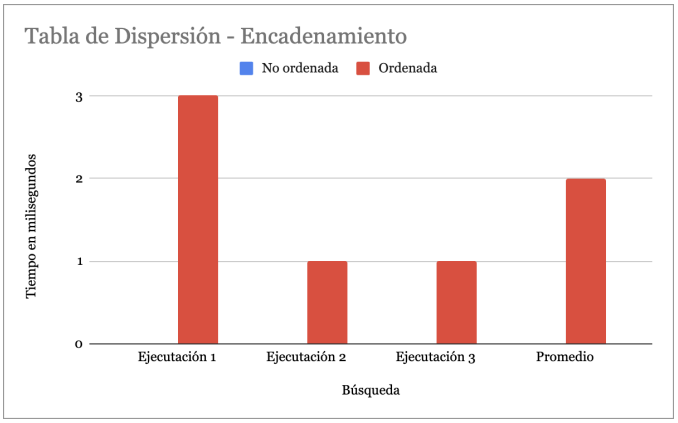


Figura 7. Tiempos de búsqueda en tabla de dispersión con encadenamiento, con inserción ordenada y no ordenada.

En la figura 7, se muestra los tiempos de búsqueda en milisegundos para una tabla de dispersión con encadenamiento (tabla hash), para elementos insertados de manera ordenada y no ordenada. Para la búsqueda de 10,000 elementos a partir de inserción ordenada, el tiempo varía un poco entre corridas, entre 1 milisegundo y 3 milisegundos, resultando un promedio de 2 milisegundos. Para la búsqueda de 10,000 elementos a partir de inserción no ordenada, el tiempo fue de 0 milisegundos para todas las ejecuciones.

En la figura 8, se muestra la cantidad de colisiones obtenidas al hacer una búsqueda de 10,000 elementos en una tabla de dispersión con encadenamiento, a partir de inserción

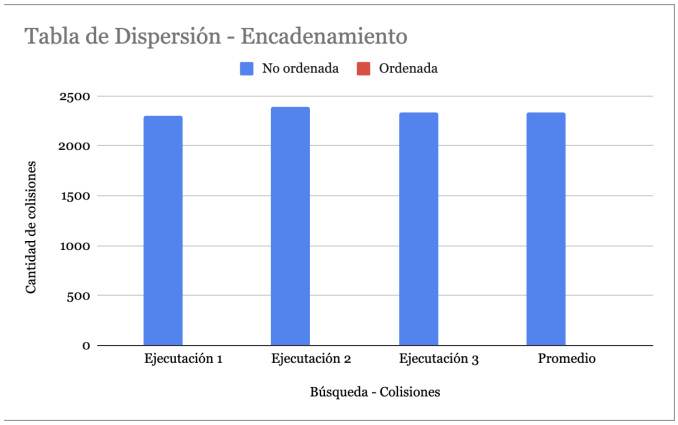


Figura 8. Colisiones en búsqueda en tabla de dispersión con encadenamiento, con inserción ordenada y no ordenada.

de elementos ordenados y no ordenados. Para la búsqueda de 10,000 elementos a partir de una inserción ordenada, la cantidad de colisiones es 0, mientras que para la búsqueda de 10,000 elementos a partir de una inserción no ordenada, la cantidad de colisiones promedio es bastante cercana a 2500, 2338 para ser exacto.

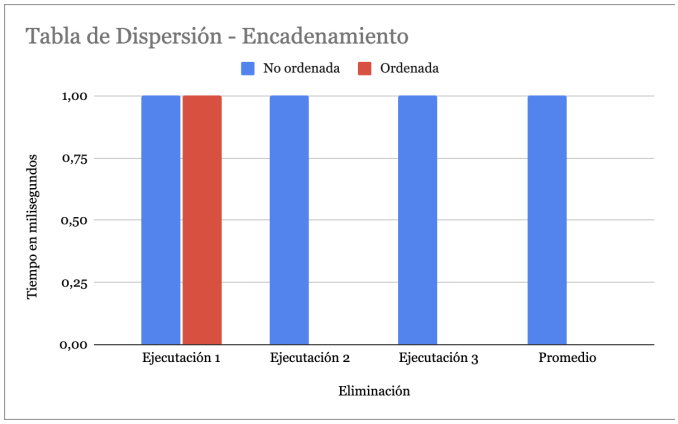


Figura 9. Tiempos de eliminación en tabla de dispersión con encadenamiento, con insersión ordenada y no ordenada.

En la figura 9, se muestra los tiempos en milisegundos de eliminación de 10,000 elementos de una tabla de dispersión con encadenamiento, a partir de inserción ordenada como inserción no ordenada. Para la eliminación de los 10,000 elementos a partir de la inserción ordenada, el promedio ronda los 0 milisegundos, solo en una ejecución alcanzó 1 milisegundo, mientras que para la eliminación de 10,000 elementos a partir de inserción no ordenada, el promedio de tiempo fue de exactamente 1 milisegundo.

En la figura 10, se muestra la cantidad de colisiones obtenidas al hacer una eliminación de 10,000 elementos en una tabla de dispersión con encadenamiento, a partir de inserción de elementos ordenados y no ordenados. Para la eliminación de 10,000 elementos a partir de una inserción ordenada, la cantidad de colisiones es 0, mientras que para la eliminación de 10,000 elementos a partir de una inserción no ordenada, la cantidad de colisiones promedio es bastante cercana a 2500,

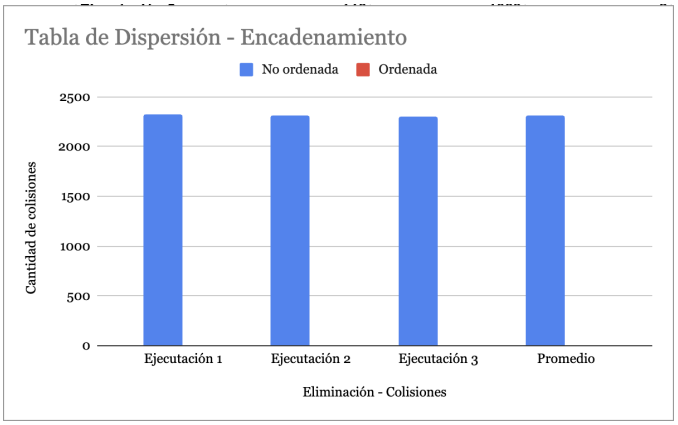


Figura 10. Colisiones en eliminación en tabla de dispersión con encadenamiento, con inserción ordenada y no ordenada.

2309 para ser exacto.

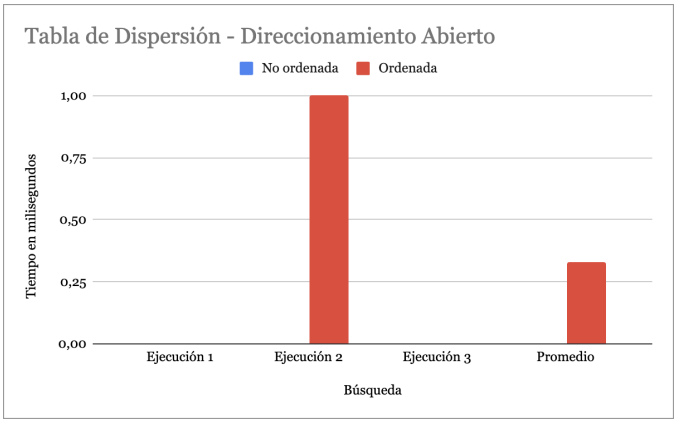


Figura 11. Tiempos de búsqueda en tabla de dispersión con direccionamiento abierto, con inserción ordenada y no ordenada.

En la figura 11, se muestra los tiempos de búsqueda en milisegundos para una tabla de dispersión con direccionamiento abierto (tabla hash), para elementos insertados de manera ordenada y no ordenada. Para la búsqueda de 10,000 elementos a partir de inserción ordenada, el tiempo varía un poco entre corridas, entre 0 milisegundos y 1 milisegundo, resultando un promedio de 0,33 milisegundos. Para la búsqueda de 10,000 elementos a partir de inserción no ordenada, el tiempo fue de 0 milisegundos para todas las ejecuciones.

En la figura 12, se muestra la cantidad de colisiones obtenidas al hacer una búsqueda de 10,000 elementos en una tabla de dispersión con direccionamiento abierto, a partir de inserción de elementos ordenados y no ordenados. Para la búsqueda de 10,000 elementos a partir de una inserción ordenada, la cantidad promedio de colisiones es de 1,095,185, mientras que para la búsqueda de 10,000 elementos a partir de una inserción no ordenada, la cantidad de colisiones promedio es bastante cercana a 90,041.

En la figura 13, se muestra los tiempos en milisegundos de eliminación de 10,000 elementos de una tabla de dispersión con direccionamiento abierto, a partir de inserción ordenada como inserción no ordenada. Para la eliminación de los 10,000

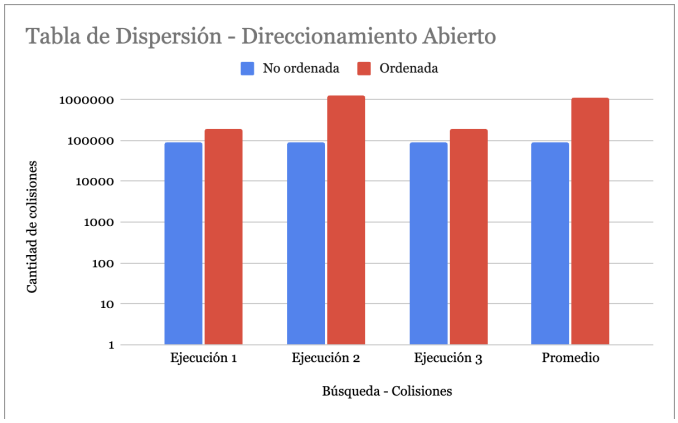


Figura 12. Colisiones en búsqueda en tabla de dispersión con direccionamiento abierto, con inserción ordenada y no ordenada.

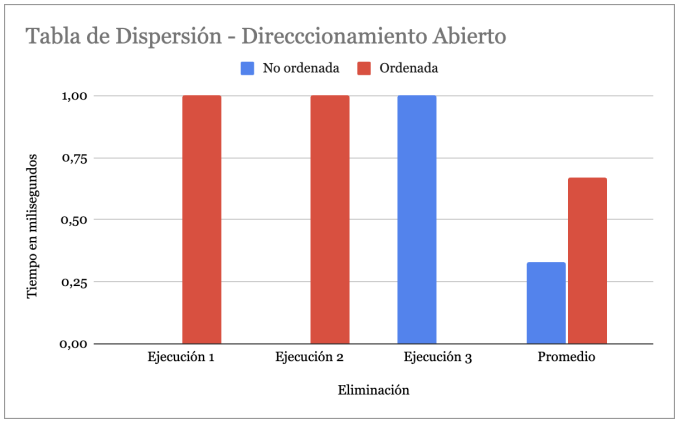


Figura 13. Tiempos de eliminación en tabla de dispersión con direccionamiento abierto, con insersión ordenada y no ordenada.

elementos a partir de la inserción ordenada, el promedio ronda los 0,67 milisegundos, con 2 ejecuciones con un 1 milisegundo y 1 con 0 milisegundos, mientras que para la eliminación de 10,000 elementos a partir de inserción no ordenada, el promedio de tiempo fue de 0,33 milisegundos, con 2 ejecuciones de 0 milisegundos y 1 de 1 milisegundo.

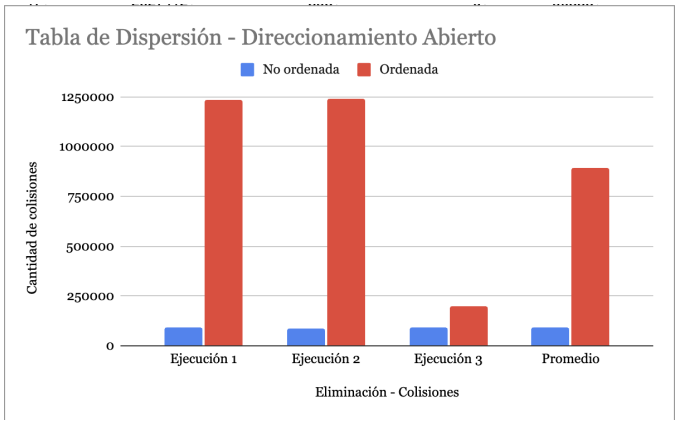


Figura 14. Colisiones en eliminación en tabla de dispersión con encadenamiento, con inserción ordenada y no ordenada.

En la figura 14, se muestra la cantidad de colisiones

obtenidas al hacer una eliminación de 10,000 elementos en una tabla de dispersión con direccionamiento abierto, a partir de inserción de elementos ordenados y no ordenados. Para la eliminación de 10,000 elementos a partir de una inserción ordenada, la cantidad de colisiones promedio es de 891,260, con 2 ejecuciones con más de 1,200,000 colisiones y una tercera de 196,808 mientras que para la eliminación de 10,000 elementos a partir de una inserción no ordenada, la cantidad de colisiones promedio es de 90,690.

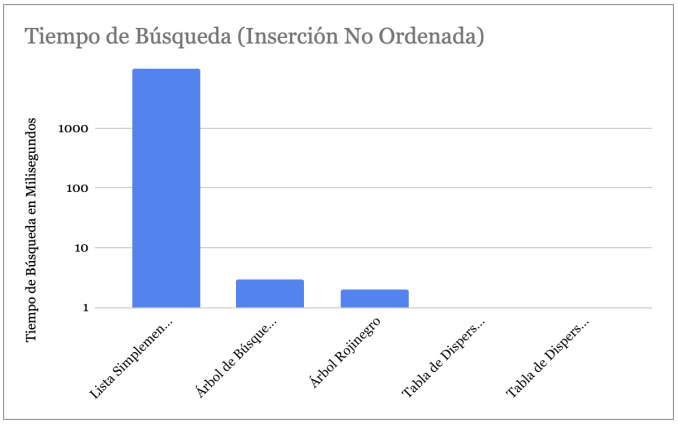


Figura 15. Tiempos de búsqueda con inserción no ordenada.

En la figura 15, se puede ver los tiempos de búsqueda en milisegundos de 10,000 elementos en cada una de las estructuras de datos (Lista simplemente enlazada, árbol de búsqueda binaria, árbol rojo-negro, tabla de dispersión con encadenamiento y tabla de dispersión con direccionamiento abierto) para inserción no ordenada. El tiempo de búsqueda para la lista simplemente enlazada es de 9,989 milisegundos, para el árbol de búsqueda binaria es de 3 milisegundos, para el árbol rojo-negro, es de 2 milisegundos y para ambas tablas de dispersión es de 0 milisegundos.

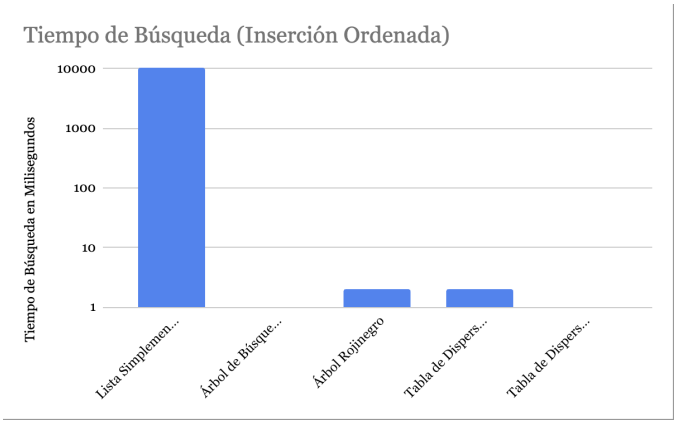


Figura 16. Tiempos de búsqueda con inserción ordenada.

En la figura 16, se puede ver los tiempos de búsqueda en milisegundos de 10,000 elementos en cada una de las estructuras de datos (Lista simplemente enlazada, árbol de búsqueda binaria, árbol rojo-negro, tabla de dispersión con encadenamiento y tabla de dispersión con direccionamiento abierto) para inserción ordenada. El tiempo de búsqueda para la lista simplemente enlazada es de 10,331 milisegundos, para el árbol de búsqueda binaria es de 1 milisegundo, para el árbol rojo-negro, es de 2 milisegundos y para la tabla de dispersión con encadenamiento es de 2 milisegundos, mientras que para la tabla de dispersión con direccionamiento abierto, es de 0,33 milisegundos.

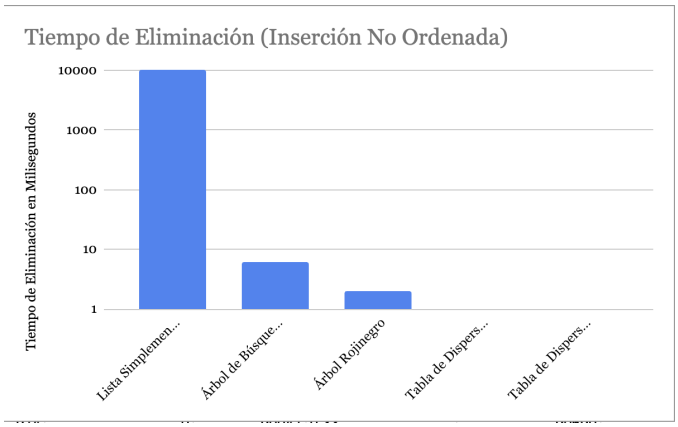


Figura 17. Tiempos de eliminación con inserción no ordenada.

En la figura 17, se puede ver los tiempos de eliminación en milisegundos de 10,000 elementos en cada una de las estructuras de datos (Lista simplemente enlazada, árbol de búsqueda binaria, árbol rojo-negro, tabla de dispersión con encadenamiento y tabla de dispersión con direccionamiento abierto) para inserción no ordenada. El tiempo de eliminación para la lista simplemente enlazada es de 10,143 milisegundos, para el árbol de búsqueda binaria es de 6 milisegundos, para el árbol rojo-negro, es de 2 milisegundos, para la tabla de dispersión con encadenamiento es de 1 milisegundo y para la tabla de dispersión con direccionamiento abierto es de 0,33 milisegundos.

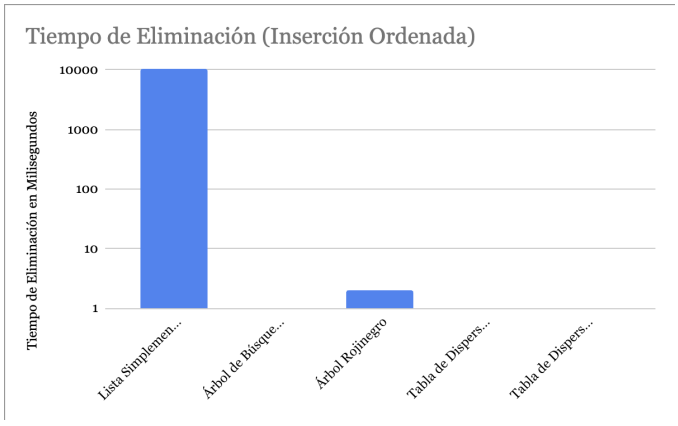


Figura 18. Tiempos de eliminación con inserción ordenada.

En la figura 18, se puede ver los tiempos de eliminación en milisegundos de 10,000 elementos en cada una de las estructuras de datos (Lista simplemente enlazada, árbol de búsqueda binaria, árbol rojo-negro, tabla de dispersión con encadenamiento y tabla de dispersión con direccionamiento abierto) para inserción ordenada. El tiempo de eliminación para la lista simplemente enlazada es de 10,428 milisegundos,

para el árbol de búsqueda binaria es de 1 milisegundo, para el árbol rojo-negro, es de 2 milisegundos y para la tabla de dispersión con encadenamiento es de 0 milisegundos, mientras que para la tabla de dispersión con direccionamiento abierto, es de 0,67 milisegundos.

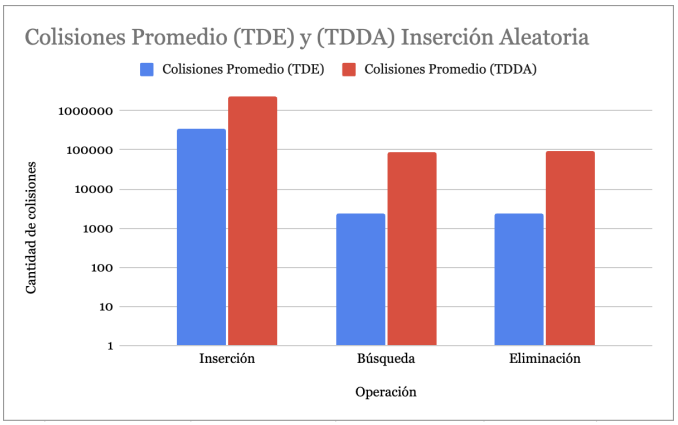


Figura 19. Colisiones promedio con inserción aleatoria en la tabla de dispersión con encadenamiento y la tabla de dispersión con direccionamiento abierto.

En la figura 19, se puede ver la cantidad de colisiones que se da en cada una de las operaciones fundamentales (inserción, búsqueda y eliminación), en la tabla de dispersión con encadenamiento y en la tabla de dispersión con direccionamiento abierto, a partir de una inserción aleatoria. Para la inserción en la tabla de dispersión con encadenamiento, se obtiene un total de 356,041, para inserción en direccionamiento abierto un total de 2,281,732, para las operación de búsqueda, para la tabla de dispersión con encadenamiento se obtiene un total de 2,338 colisiones, mientras que para la tabla de dispersión con direccionamiento se obtiene 90,041 colisiones y por último para la operación de eliminación, para la tabla con encadenamiento se obtiene un total de 2,309 colisiones, mientras que parala tabla de direccionamiento abierto, se obtiene un total de 90,690 colisiones.

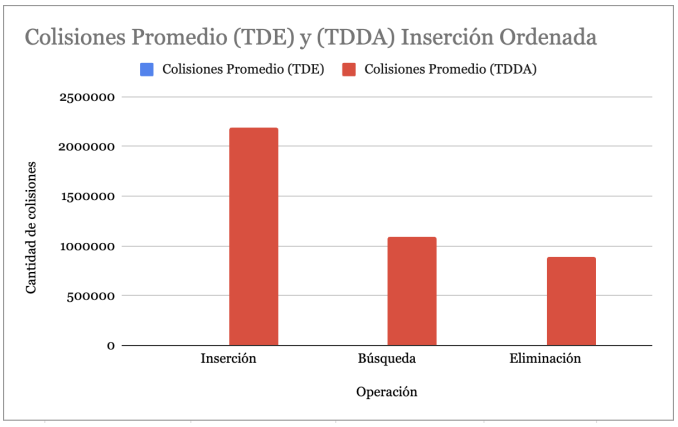


Figura 20. Colisiones promedio con inserción ordenada en la tabla de dispersión con encadenamiento y la tabla de dispersión con direccionamiento abierto.

En la figura 20, se puede ver la cantidad de colisiones que se da en cada una de las operaciones fundamentales (inserción, búsqueda y eliminación), en la tabla de dispersión con encadenamiento y en la tabla de dispersión con direccionamiento abierto, a partir de una inserción ordenada. Para la tabla de sipersión con encadenamiento, las 3 operaciones dan 0 colisiones, mientras que para la tabla de dispersión con encadenamiento abierto se obtiene 2,187,562 colisiones para inserción, 1,095,185 colisiones para la operación de búsqueda y 891,260 para la operación de eliminación.

IV. Discusión

Comportamiento global de las estructuras de datos

En términos generales, las estructuras de datos mostraron rendimientos bastante esperables, las listas simplemente enlazadas mostraron un comportamiento consistente con la teoría. Las inserciones al inicio son muy rápidas por operación, pero al no permitirse la inserción de elementos duplicados, se obliga a hacer una operación de búsqueda en la lista, la cual tiene un costo n, elevando así el tiempo de inserción de la lista. Las búsquedas y eliminaciones presentan complejidad lineal O(n), y los tiempos medidos reflejan esta expectativa, mostrando que el orden de inserción (aleatorio u ordenado) no influye significativamente en la eficiencia de estas operaciones. Las pequeñas variaciones observadas se explican por la distribución de los elementos y la implementación particular de cada prueba.

Los árboles binarios de búsqueda y los árboles rojinegros exhiben un patrón claramente más eficiente para búsquedas y eliminaciones comparados con las listas. En árboles binarios de búsqueda, el uso de fast Insert (implementación solicitada, que mantiene el árbol balanceado) permitió que las inserciones ordenadas fueran muy rápidas, mientras que las aleatorias resultaron moderadas, reflejando la dependencia de la altura del árbol en el rendimiento. Por su parte, los árboles rojinegros mantienen un auto balanceo constante, garantizando alturas logarítmicas y tiempos estables en todas las operaciones, aunque con un pequeño tiempo perdido adicional por las rotaciones y recoloreos necesarios para mantener las propiedades de balance.

Las tablas de dispersión o comúnmente conocidas como tablas hash, tanto con encadenamiento como con direccionamiento abierto (double hashing), demostraron ser las estructuras más rápidas en promedio, con búsquedas y eliminaciones prácticamente O(1). El encadenamiento mostró colisiones moderadas, mientras que la tabla con direccionamiento abierto reportó millones de intentos fallidos debido al alto factor de carga aunque esto no afectó significativamente los tiempos reales de operación. La eficiencia de estas estructuras refleja la importancia de un buen hash y de estrategias que minimicen clustering, como el double hashing frente al sondeo cuadrático.

Nota: Esto nació a raíz de que se había hecho en principio una implementación de la tabla de dispersión con direccionamiento abierto usando sondeo cuadrático, pero la cantidad de colisiones se disparaba por completo, a más de 600 millones de colisiones para inserción de 1.000.000 elementos de manera ordenada, por lo que se decidió cambiar por double hashing, sin embargo, al investigar un poco, se pudo deducir que mucho tenía que ver el factor de carga tan alto y el clustering (muchas posiciones adyacentes ocupadas).

Finalmente, al comparar globalmente todas las estructuras, se observa claramente la jerarquía esperada, listas enlazadas como la opción más lenta, los árboles de búsqueda binaria y los árboles rojinegros con tiempos logarítmicos más bajos, y tablas hash como las más eficientes. La diferencia principal entre escenarios de inserción aleatoria y ordenada recae en la forma de construcción de los árboles de búsqueda binaria (fastInsert), que mejora notablemente el rendimiento de búsquedas y eliminaciones. En general, los resultados concuerdan con la teoría, y las variaciones prácticas se explican por detalles de implementación y distribución de datos, confirmando que la selección de la estructura debe considerar tanto el patrón de acceso como la estrategia de construcción.

Comparación entre tabla de dispersión con encadenamiento y tabla de dispersión con direccionamiento abierto

Al comparar las dos tablas hash, se observa que tanto tabla de dispersión con encadenamiento como tabla de dispersión con direccionamiento abierto con doble hashing presentan un desempeño muy eficiente en términos de tiempo de operación, aunque con diferencias interesantes en cómo cada una gestiona las colisiones. En la tabla con encadenamiento, las inserciones son rápidas, en inserciones aleatorias como ordenadas, ya que la función hash distribuye de forma bastante eficiente los elementos. Por otra parte la tabla con direccionamiento abierto muestran igualmente inserciones rápidas, principalmente en secuencias ordenadas, pero con una cantidad enorme de colisiones, eso si, estas colisiones no afectan significativamente el tiempo real de ejecución, pero reflejan cómo la estrategia de resolución de colisiones impacta el conteo interno de operaciones.

En las búsquedas y eliminaciones se mantiene esta diferencia en comportamiento. En la tabla con encadenamiento, las búsquedas son consistentes y rápidas, con pocas colisiones, y las eliminaciones se ejecutan de forma directa y rápida ajustando los punteros de la lista. Direccionamiento abierto, aunque muy rápido en tiempo real para búsquedas y eliminaciones, muestra un número elevado de colisiones. Esto evidencia que el rendimiento en tiempo no siempre se correlaciona con la cantidad de colisiones, y que la eficiencia práctica depende de cómo la estructura maneje los elementos eliminados y la carga de la tabla.

Además, es importante mencionar que el orden de inserción tiene un efecto distinto en cada estructura, por ejemplo, la tabla con encadenamiento es bastante robusta frente a variaciones en el orden, ya que la distribución uniforme de la función hash mantiene las celdas equilibradas. En direccionamiento abierto, el orden puede influir en la cantidad de intentos necesarios, sobre todo cuando el factor de carga es alto y se acumulan marcas DELETED, lo que incrementa la cantidad de intentos para encontrar una celda disponible. En términos teóricos, ambas tablas alcanzan complejidad O(1), pero la tabla con encadenamiento demuestra un poco de mayor estabilidad frente a patrones de inserción, mientras que direccionamiento abierto depende de un hashing eficiente y de un manejo cuidadoso de las eliminaciones para mantener un rendimiento consistente.

Comparación entre los dos árboles binarios

Al comparar el árbol de búsqueda binaria con el árbol rojinegro, se observa que ambos mantienen tiempos de búsqueda y eliminación significativamente más bajos que las listas simples, gracias a su altura logarítmica. Sin embargo, esto no se cumple en el arbol de busqueda binaria, cuando se insertan elementos en algún orden, ya sea creciente o decreciente, donde el árbol quedaría completamente desbalanceado y con altura n, sin embargo, el uso de fastInsert para la inserción ordenada permitió construir un árbol prácticamente balanceado en tiempo lineal, lo que se refleja en búsquedas extremadamente rápidas y eliminaciones eficientes. Por el contrario, en el escenario aleatorio, el árbol de búsqueda binaria muestra tiempos algo más altos debido a que el balance depende del azar, es decir, no hay una garantía de que el árbol va a quedar balanceado, sin embargo, incluso en este caso, las operaciones siguen siendo considerablemente más rápidas que en listas, permitiendo ver como el rendimiento de este depende directamente de su proceso de construcción.

El árbol rojinegro, por su parte, ofrece un comportamiento consistente en ambos escenarios, aleatorio y ordenado, debido a su auto-balanceo característico mediante rotaciones y recoloreos. Aunque las inserciones son algo más costosas que en el árbol de búsqueda binaria, este tiempo extra, hace que la altura del árbol nunca sea mayor a logn, garantizando búsquedas y eliminaciones rápidas y estables. En conjunto, la comparación muestra que mientras el árbol de búsqueda binaria puede superar al árbol rojinegro en escenarios controlados con fastInsert, el árbol rojinegro brinda robustez y consistencia sin depender de estrategias externas de balanceo, lo que lo hace más predecible frente a distintos patrones de inserción.

Influencia del orden de inserción según estructura

El orden de inserción, aunque no lo parezca, puede ser un factor clave en el rendimiento de las estructuras de datos analizadas en algunas de sus operaciones esenciales, sin embargo, no todos se ven influenciados por esto. Iniciando por la lista simplemente enlazada, de acuerdo a los resultados obtenidos, es seguro decir que muestran prácticamente ningún efecto del orden, los tiempos de búsqueda, eliminación e inserción dependen más del tamaño de n.

En el caso de los árboles rojinegros, la situación cambia por completo, ya que cuando se inserta una serie de elementos siguiendo un orden definido, creciente o decreciente, lo que se genera realmente es un árbol completamente desbalanceado, que realmente se puede ver como una lista, por lo que pasaría de tener operaciones con costos logarítmicos a tener costos lineales, debido a que se da su peor caso. Los árboles rojinegros y las tablas de dispersión (encadenamiento y direccionamiento abierto) son mucho menos sensibles al orden, ya que los árboles rojinegros se auto-balancean y las tablas de dispersión dependen principalmente de la función hash y factor de carga, pero en general no se ven muy afectados, al menos en tiempos, porque de acuerdo a los resultados, no se puede decir lo mismo en cuanto a colisiones.

Relación con la teoría

Los resultados en general confirman las complejidades esperadas, tiempos lineales para listas o árboles binarios desbalanceados, tiempos logarítmicos para árboles balanceados y tiempos constantes para las tablas de dispersión. Los resultados obtenidos para la lista simplemente enlazada se alinean con lo expuesto en la teoría. Tal como se observó, la inserción al inicio fue eficiente al ser una operación de costo constante, sin embargo, su tiempo se dispara a partir de la limitante de que no se permiten elementos repetidos, lo que obliga a hacer una búsqueda. Y es que la búsqueda y eliminación resultaron más costosas debido a que requieren recorrer secuencialmente los nodos hasta encontrar el elemento, con complejidad lineal. Esto coincide con la afirmación del texto, de que "...para buscar en una lista de objetos, el procedimiento LIST-SEARCH toma tiempo (n) en el peor caso, ya que podría tener que recorrer la lista completa. " (Cormen, Leiserson, Rivest, Stein, 2022, p.260). Por ello, los tiempos elevados observados para inserción, búsqueda y eliminación son coherentes con la complejidad (n), ya que todas dependen en alguna medida de la operación de búsqueda.

En el caso de los árboles de búsqueda binaria y los árboles rojinegros, los datos experimentales reflejan fielmente la teoría sobre esta estructura. El árbol de búsqueda binaria mostró un desempeño acorde a su forma, cuando el árbol se construyó con inserciones aleatorias, su altura se mantuvo moderada, logrando búsquedas en torno a O(log n), pero al aplicar un método de construcción balanceada (fastInsert), los tiempos mejoraron significativamente al reducirse la altura efectiva del árbol, independientemente de si la inserción era ordenada, lo cual teóricamente lleva al peor caso. Esto concuerda con la definición de árbol de búsqueda binaria, donde "...las operaciones básicas en un árbol binario de búsqueda requieren un tiempo proporcional a la altura del árbol" (Cormen, Leiserson, Rivest, Stein, 2022, p.312).

Por su parte, el ARN conservó siempre un rendimiento estable debido a su propiedad de auto-balanceo, manteniendo su altura O(log n) independientemente del orden de inserción, gracias a rotaciones y recoloreos que preservan el balance, tal y como menciona Cormen estos "...son uno de los varios esquemas de árboles de búsqueda que se mantienen balanceados para garantizar que las operaciones básicas de conjuntos dinámicos tomen tiempo logarítmico en el peor caso." (Cormen, Leiserson, Rivest, Stein, 2022, p.331). Así, los tiempos constantes observados para búsquedas y eliminaciones en árbol rojinegro demuestran precisamente el beneficio del balance automático frente a un árbol de búsqueda binaria que depende del patrón de inserción.

Finalmente, los resultados obtenidos para las tablas de dispersión, tanto con encadenamiento como con direccionamiento abierto, se ajustan plenamente al comportamiento esperado. En el encadenamiento, las operaciones se mantuvieron bastante rápidas, confirmando la eficiencia constante promedio, cuando la función distribuye de forma adecuada los valores, aunque se puede ver que existe una tendencia a que la tabla de dispersión con direccionamiento abierto, sea un poco más rápida.

Esto se puede justificar con el hecho de que si en encadenamiento, se dan varias colisiones, lo que hace es que se anexan en la respectiva lista, por lo que "...buscar un elemento en una tabla hash puede tardar tanto como buscar un elemento en una lista enlazada." (Cormen, Leiserson, Rivest, Stein, 2022, p.272).

Limitaciones

Algunas limitaciones a considerar son que si bien es cierto para todas las estructuras, se busca insertar 1 millón de elementos, y buscar y eliminar 10.000, los datos insertados en cada estructura no son iguales, por lo que puede haber pequeñas variaciones en la eficiencia a raíz de esto, la cantidad de colisiones puede aumentar o disminuir y los tiempos de búsqueda pueden reflejar mejoras significativas a raíz de esto, dependiendo de que tan profundo se deba ir en la estructura buscando el elemento hasta encontrarlo, por lo que el análisis comparativo, muestra una visión o idea bastante coherente con lo esperado, este hecho de someter las estructuras a los mismos datos, podría ser un aspecto de mejora a considerar.

V. Conclusiones

A partir de los resultados obtenidos, se pudo observar un comportamiento bastante coherente con lo que se esperaba desde la teoría para cada una de las estructuras evaluadas. En general, las diferencias en los tiempos de ejecución reflejaron justamente las fortalezas y limitaciones propias de cada estructura, pero también dejaron en evidencia el peso que tienen ciertos detalles de implementación en el rendimiento final.

En el caso de las operaciones de inserción, búsqueda y eliminación, las estructuras balanceadas y las tablas de dispersión fueron las que mostraron un desempeño más consistente y eficiente, mientras que la lista simplemente enlazada mantuvo los tiempos más altos, tal y como se anticipaba por su naturaleza lineal. Uno de los resultados más notorios fue el impacto positivo que tuvo el método utilizado para construir el Árbol Binario de Búsqueda. Cuando los elementos fueron insertados en orden utilizando la técnica de construcción balanceada, el árbol de búsqueda binaria logró mejorar de forma considerable sus tiempos de búsqueda y eliminación, siendo incluso más rápido que muchos de los otros casos. Una curiosidad es que, en este escenario particular, el árbol llegó a rendir muy por encima de lo esperado e inclusive por encima del rendimiento del mismo árbol, en la versión de inserción aleatoria, sin embargo, esto se puede justificar a partir de la misma aleatoriedad, ya que no existe una garantía de balance del árbol, sino que igualmente se puede generar un árbol bastante desbalanceado, en el cual la altura impacta negativamente en las operaciones de búsqueda y eliminación.

Los árboles rojinegros se mantuvieron estables en todos los escenarios, sin diferencias relevantes entre inserciones ordenadas o aleatorias. Esto confirma el beneficio esperado del auto-balanceo, restando así importancia al orden de llegada de los elementos y manteniendo en cualquier escenario, un comportamiento muy rápido, pero sobre todo muy consistente. Por otro lado, las tablas de dispersión también reflejaron el comportamiento previsto, con tiempos muy bajos y consistentes en la mayoría de operaciones. El uso de double hashing

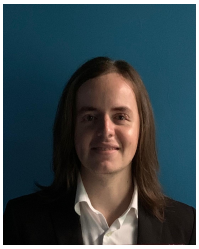
para el direccionamiento abierto permitió reducir de manera significativa los problemas de colisiones y bajo rendimiento que se habían observado con el sondeo cuadrático. Aun así, dado el alto factor de carga de alrededor del 95

Finalmente, al comparar los escenarios con inserción ordenada e inserción aleatoria, fue claro que no todas las estructuras se ven afectadas de la misma manera y que realmente afecta menos de lo que se podría pensar en un principio. En particular, la forma en que se construyó el árbol de búsqueda binaria para el escenario de inserción ordenada, marcó una diferencia importante, ya que, de no ser así, acá se habría reflejado una importante baja en el rendimiento, mientras que en otras estructuras como la lista, el árbol rojinegro o las tablas de dispersión, los cambios entre un escenario y otro fueron mínimos o poco significativos, pero en este caso más por la naturaleza de las estructuras que de la construcción de la misma, como sí lo fue en el árbol de búsqueda binaria. Esto confirma que el orden de los datos sí puede tener un impacto, pero su efecto depende en gran medida del tipo de estructura y de si esta posee mecanismos internos o externos para preservar el balance.

En términos generales, los resultados obtenidos fueron consistentes con lo que se esperaba desde la teoría y permitieron analizar de forma comparativa el comportamiento práctico de cada estructura y responder a las interrogantes planteadas en los objetivos del análisis. Las observaciones realizadas aportan una idea sobre cuándo podría convenir utilizar una estructura sobre otra, pero no es como que se pueda afirmar categóricamente resultados concluyentes o hacer afirmaciones sobre cuál es mejor o peor.

REFERENCIAS

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2022). Introduction to algorithms (4th ed.). MIT Press.



Jean Aguilar Mena Soy una persona bastante alegre, que le gusta mucho aprender cosas nuevas. Me gusta mucho aprender sobre tecnología y soy un amante de la música. Me gusta hablar y conocer personas, porque me dan una perspectiva diferente, lo cual, en mi caso, siempre es bienvenido.