Análisis Comparativo del Desempeño de Estructuras de Datos en Operaciones de Inserción y Búsqueda

Camila Rodríguez Águila - C36624

CI-0116 Análisis de Algoritmos y Estructuras de Datos - I-2025

Correo Electrónico: camila.rodriguezaguila@ucr.ac.cr

Abstract

Este estudio presenta una evaluación del rendimiento de cuatro estructuras de datos: lista enlazada, árbol binario de búsqueda, tabla hash y árbol rojo-negro. Se analizaron las operaciones de búsqueda y eliminación bajo dos patrones de entrada distintos: datos aleatorios y datos ordenados secuencialmente. Los experimentos revelan diferencias significativas en el comportamiento temporal de cada estructura, donde la tabla hash demostró el mejor rendimiento general con complejidad prácticamente O(1), mientras que el árbol binario simple exhibió la mayor sensibilidad al patrón de entrada. Los resultados confirman las predicciones teóricas de complejidad asintótica y revelan que la elección de estructura de datos debe considerar tanto el patrón de entrada como las operaciones predominantes en la aplicación específica.

Palabras clave: estructuras de datos, análisis empírico, complejidad computacional, rendimiento, búsqueda, eliminación

1. Introducción

Según TechTarget (2024) “Una estructura de datos es un formato especializado para organizar, procesar, recuperar y almacenar datos”, estas establecen el fundamento sobre el cual se construyen algoritmos eficientes en computación. Mientras que el análisis teórico de complejidad asintótica provee cotas sobre el comportamiento esperado de estas estructuras, las condiciones de ejecución pueden revelar diferencias significativas respecto a las predicciones teóricas. La comprensión de estas diferencias es crucial para la toma de decisiones informadas en el diseño de sistemas computacionales.

El objetivo de esta investigación práctica es comparar los tiempos de ejecución de cuatro estructuras de datos fundamentales y comparar el cómo estos se relacionan con las cotas teóricas

establecidas en la teoría algorítmica. Las estructuras seleccionadas representan diferentes paradigmas de organización de datos: acceso secuencial (lista enlazada), organización jerárquica (árboles binarios), dispersión por función hash (tabla hash) y auto-balanceo (árbol rojo-negro).

La evaluación del rendimiento bajo diferentes patrones de entrada es sorpresivamente relevante, ya que las aplicaciones reales presentan distribuciones de datos dispersos. Cormen establece que el comportamiento de estructuras como los árboles binarios de búsqueda puede degradarse significativamente cuando los datos de entrada siguen patrones específicos, transformando su complejidad promedio de O(log n) a O(n) en el peor caso. Esta investigación busca cuantificar esos efectos mediante mediciones realizadas bajo las mismas condiciones.

La justificación de este estudio se establece en la necesidad de validar experimentalmente las predicciones teóricas y proporcionar datos cuantitativos que permitan a los desarrolladores seleccionar la estructura más apropiada según el contexto de aplicación. La investigación se organiza presentando primero la metodología experimental, seguida de los resultados obtenidos, su discusión en el contexto de la teoría existente, y finalmente las conclusiones derivadas del análisis.

2. Metodología

2.1 Estructuras de Datos Evaluadas

Se implementaron cuatro estructuras de datos: lista enlazada simple, árbol binario de búsqueda sin balanceo, tabla hash con resolución de colisiones por encadenamiento, y árbol rojo-negro. La selección de estas estructuras permite evaluar el impacto de características como el auto-balanceo y la dispersión por función hash al someterse a diferentes entradas.

2.2 Operaciones Analizadas

Las operaciones evaluadas fueron búsqueda y eliminación de elementos. Para la búsqueda, se midió el tiempo requerido para localizar elementos en la estructura. Para la eliminación, se registró el tiempo total incluyendo la localización y eliminación del elemento, en el caso de las tablas hash. Ambas operaciones suelen ser utilizadas con frecuencia en el análisis de estructuras de datos y las mismas permiten evaluar el rendimiento.

2.3 Generación y Patrones de Datos

Se utilizaron conjuntos de 1 000 000 de elementos enteros generados bajo dos patrones distintos. Se generaron e números enteros aleatorios utilizando una distribución uniforme con el generador pseudoaleatorio mt19937 (Mersenne Twister). Este método garantiza una buena dispersión de valores dentro del rango especificado, mientras que los datos ordenados consistieron en secuencias crecientes comenzando desde 0. Estos diferentes patrones permiten evaluar el impacto del patrón de entrada en estructuras sensibles al orden de inserción.

2.4 Procedimiento de Medición

El experimento consistió en tres ejecuciones independientes para cada combinación de estructura, operación y patrón de datos. Los tiempos se midieron en segundos utilizando marcas de tiempo de alta precisión. El promedio aritmético de las tres mediciones se utilizó como valor representativo de los datos. Por lo tanto, en este reporte, dicho promedio será considerado como el valor de referencia para realizar las comparaciones.

2.5 Ambiente Experimental Controlado

Durante la ejecución de los experimentos, el sistema se mantuvo en estado dedicado, evitando procesos concurrentes que pudieran cambiar algo en las mediciones. Se implementaron rutinas de limpieza de memoria entre experimentos para garantizar condiciones iniciales consistentes.

2.6 Ambiente de Ejecución

Los experimentos se ejecutaron en un sistema con procesador Intel Core i5, 16 GB de memoria RAM DDR4 y sistema operativo Windows 10. Las implementaciones se desarrollaron en lenguaje C++ utilizando el entorno WSL (Windows Subsystem for Linux) y el compilador GCC (GNU Compiler Collection).

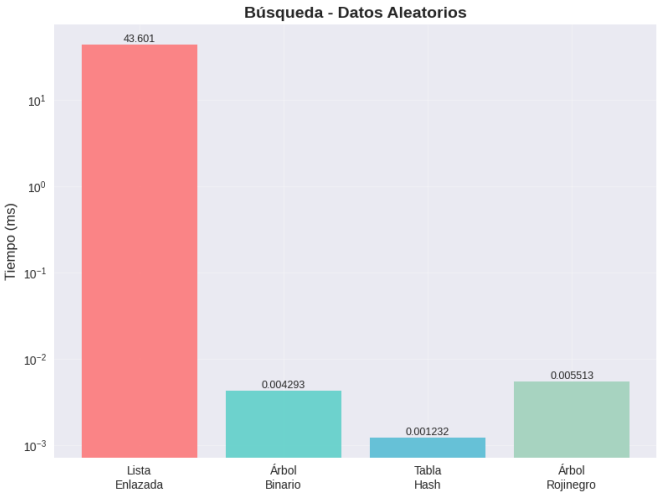
3. Resultados

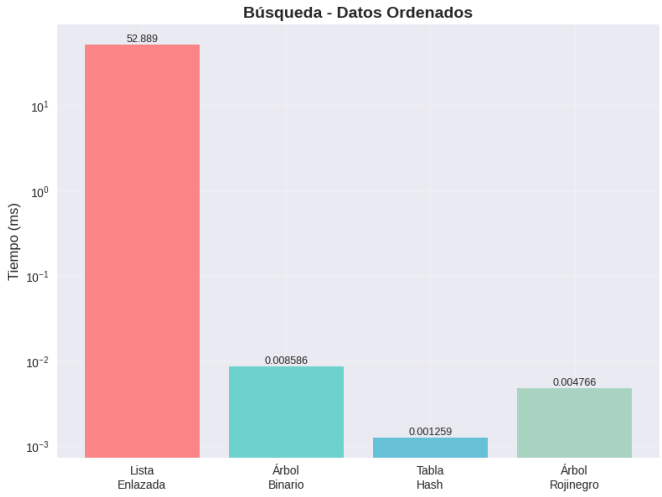
3.1 Tiempos de Ejecución por Operación

La Tabla I presenta los tiempos promedio de ejecución para todas las configuraciones experimentales. Los resultados muestran variaciones de varios órdenes de magnitud entre diferentes estructuras, con rangos desde 0.0001 milisegundos hasta más de 50 milisegundos para las mismas operaciones.

Tabla I: Tiempos Promedio de Ejecución (milisegundos)

Estructura	Operación	Datos Aleatorios	Datos Ordenados
Lista Enlazada	Búsqueda	43.601	52.889
Lista Enlazada	Eliminación	46.848	50.377
Árbol Binario	Búsqueda	0.0004293	0. 0.008586
Árbol Binario	Eliminación	0.004226	0. 0.0084524
Tablas Hash	Búsqueda	0.001232	0.001259
Tablas Hash	Eliminación	0.001532	0.001422
Árbol Rojinegro	Búsqueda	0.005513	0.004766
Árbol Rojinegro	Eliminación	0.004810	0.006582





3.2 Análisis de Rendimiento Relativo

La Tabla II muestra el ranking de rendimiento para operaciones de búsqueda bajo diferentes patrones de datos. Se observan reordenamientos significativos en la clasificación según el patrón de entrada, particularmente para el árbol binario simple.

Tabla II: Ranking de Rendimiento - Operación de Búsqueda

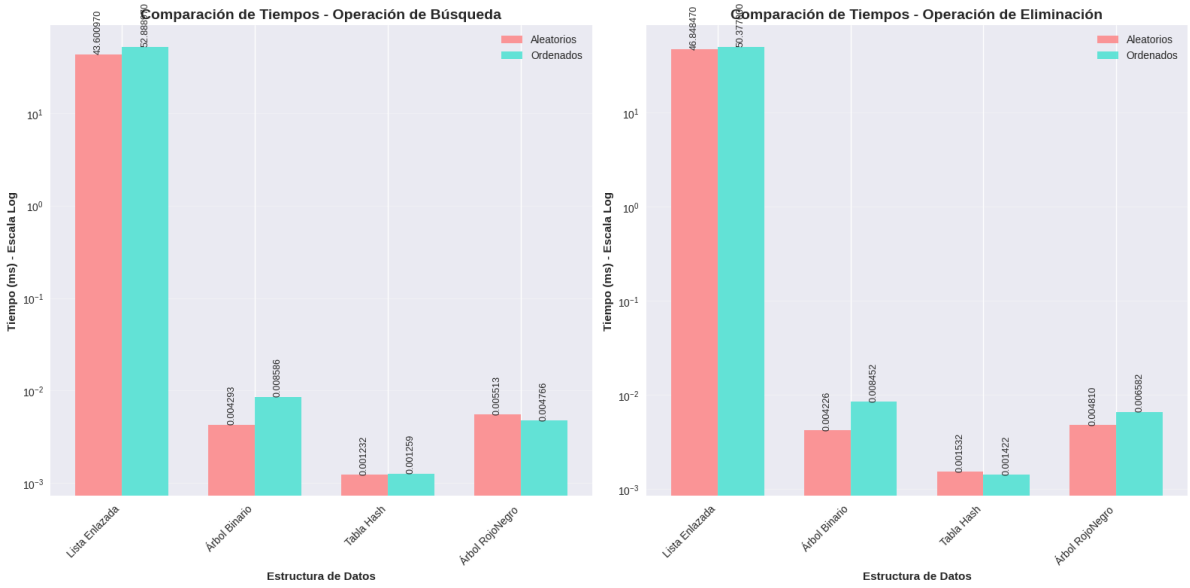
Posición	Datos Aleatorios	Tiempo (s)	Datos Ordenados	Tiempo (s)
1	Tabla Hash	0.001232	Tablas Hash	0.001259
2	Árbol Binario	0.004293	Árbol Rojo-Negro	0.004766
3	Árbol Rojo-negro	0.005513	Árbol Binario	0.008586
4	Lista Enlazada	43.601	Lista Enlazada	52.889

3.3 Impacto del Patrón de Entrada

La Tabla III cuantifica el impacto del orden de los datos en el rendimiento de cada estructura, expresado como porcentaje de cambio respecto al caso de datos aleatorios.

Tabla III: Impacto del Patrón de Datos en el Rendimiento

Estructura	Operación	Cambio Porcentual
Lista Enlazada	Búsqueda	+21.3%
Lista Enlazada	Eliminación	+7.5%
Árbol Binario	Búsqueda	+96.2%
Árbol Binario	Eliminación	+95.9%
Tabla Hash	Búsqueda	+2.2%
Tabla Hash	Eliminación	-7.2%
Árbol Rojo-Negro	Búsqueda	-13.5%
Árbol Rojo-Negro	Eliminación	+36.8%

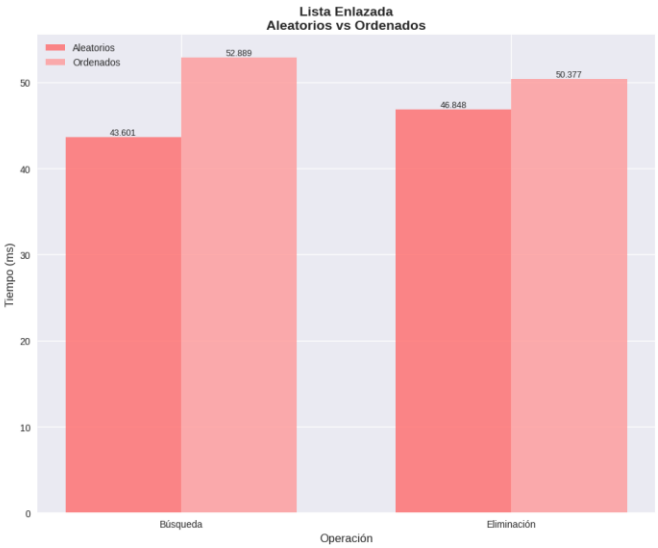


4. Discusión

4.1 Análisis de la Lista Enlazada

Se observa que, bajo las condiciones del experimento, la lista enlazada muestra un desempeño coherente con la complejidad teórica O(n) para las operaciones de búsqueda y eliminación. Los tiempos obtenidos reflejan la necesidad de realizar recorridos secuenciales ,totales o parciales, sobre la estructura, como es de esperarse en este tipo de implementación.

En promedio, los datos aleatorios resultaron más rápidos que los ordenados, con 43.6 segundos frente a 52.8 segundos en búsqueda, y 48.8 segundos frente a 50.4 segundos en eliminación. Esta diferencia, se nota aún más en la búsqueda (un 21.3% más lenta en el caso ordenado), aunque intuitivamente se pensaría que datos ordenados deberían procesarse más eficientemente, los patrones de acceso consecutivo pueden, en ciertos casos,

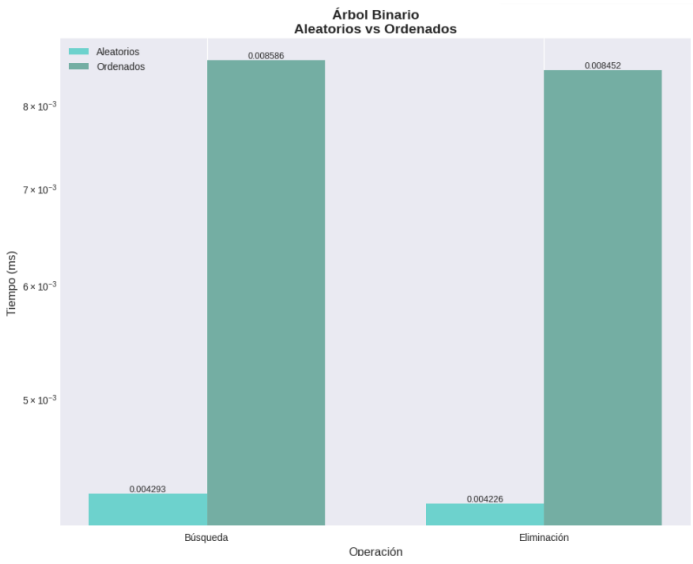


generar conflictos en caché o menos aprovechamiento de la localidad temporal, haciendo que el acceso aleatorio resulte más favorable (Intel, 2015).

Personalmente, durante el experimento surgió una duda que siempre había tenido en mente: ¿por qué la RAM es eficiente si es de accesos aleatorios? El hecho de que los datos distribuidos aleatoriamente obtuvieran mejores tiempos que los ordenados me llevó a investigar más sobre este comportamiento. Después de investigar un poco, entendí mejor por qué la RAM se denomina Random Access Memory. A diferencia de dispositivos de acceso secuencial, como las cintas magnéticas, la RAM permite acceder directamente a cualquier dirección de memoria sin tener que recorrer las anteriores, lo que le otorga tiempos de acceso uniformes. Esta característica significa que, en general, los patrones de acceso aleatorio no se ven penalizados como ocurre en el almacenamiento secundario, donde la localización física de los datos sí impacta el rendimiento.

De igual forma, aunque las listas enlazadas no aprovechan el acceso aleatorio de la RAM en sentido estricto (ya que siguen punteros), este experimento me permitió despejar esa duda personal sobre la velocidad de los accesos aleatorios en memoria principal y comprender mejor cómo su arquitectura impacta ciertos patrones de acceso.

4.2 Análisis de Árboles Binarios de Búsqueda



De acuerdo a los datos obtenidos del Árbol Binario, donde los datos aleatorios muestran tiempos de 0.004293 ms en búsqueda frente a 0.008586 ms con datos ordenados (aproximadamente el doble de rápido), este comportamiento se explica por la *degeneración estructural del árbol*.

Cuando se insertan datos en orden secuencial, se obtiene un BST donde cada nuevo elemento se convierte siempre en el hijo derecho del nodo anterior. Esta entrada transforma efectivamente el árbol en una lista enlazada, perdiendo completamente las ventajas logarítmicas de la estructura arbórea. La inserción en un árbol degenerado es O(n) porque el árbol

es esencialmente una lista enlazada. Por el contrario, los datos aleatorios permiten que el árbol mantenga una estructura más equilibrada, donde los elementos se distribuyen de manera más uniforme entre los subárboles izquierdo y derecho. El tiempo de inserción será O(n) en un árbol degenerado o Θ(log n) en un árbol binario perfecto; generalmente, mientras menor sea la altura del árbol, mejor será el rendimiento.

Los resultados empíricos corroboran el análisis teórico: cuando los datos de entrada están ordenados, un árbol binario de búsqueda (BST) degenera en una lista enlazada con altura Θ(n), donde nn es el número de nodos. Este escenario representa el peor caso para las operaciones de búsqueda y eliminación, ya que su tiempo de ejecución se vuelve O(n).

Por el contrario, con datos aleatorios, el BST tiende a mantenerse balanceado en altura esperada O(logn), preservando así la eficiencia logarítmica teórica O(logn) por operación). Esta diferencia estructural explica la discrepancia observada en los tiempos de ejecución:

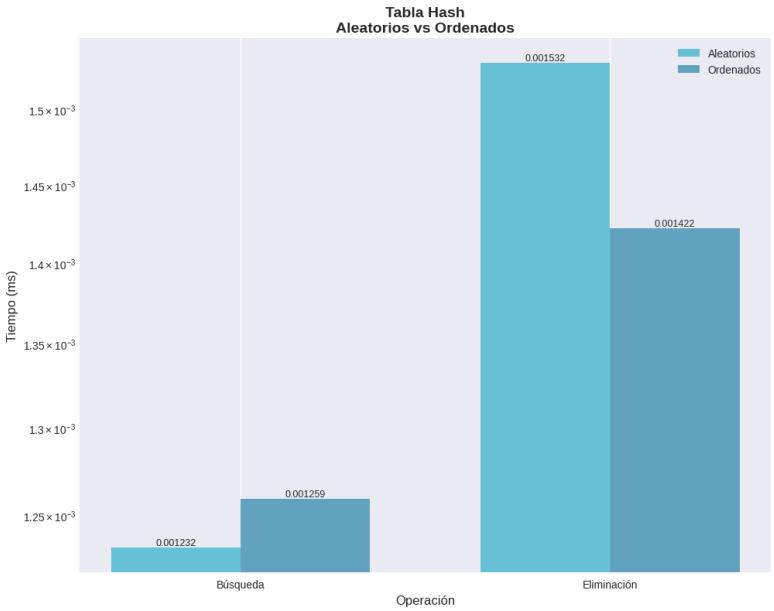
- Datos ordenados: O(n) (degeneración).
- Datos aleatorios: O(logn) (altura balanceada).

Como se demuestra en la Tabla III, el BST exhibe un incremento en el tiempo de operaciones con datos ordenados, mientras que estructuras como el árbol rojo-negro (que garantizan balanceo mediante invariantes) mantienen complejidad O(logn) en todos los casos.

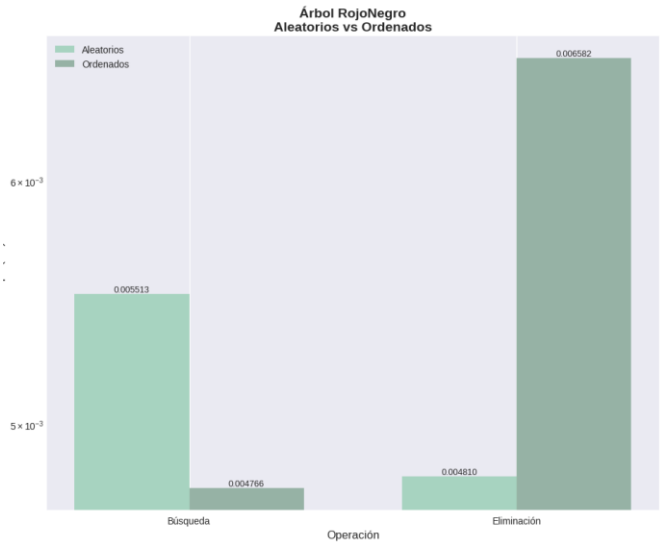
4.3 Análisis de Tablas Hash

Las tablas hash demuestran el comportamiento más estable ante diferentes patrones de entrada, con variaciones menores al 10% entre datos aleatorios y ordenados. Los tiempos registrados (aproximadamente 0.001 ms) son consistentes con la complejidad teórica O(1) para operaciones de búsqueda y eliminación. Las tablas hash ofrecen tiempos de acceso prácticamente constantes independientemente del patrón de entrada, lo cual se confirma en los resultados experimentales.

La resistencia de las tablas hash ante diferentes patrones de datos las convierte en la opción más predecible para aplicaciones donde la variabilidad del rendimiento debe minimizarse. Su superioridad absoluta en términos de velocidad (35 veces más rápida que la lista enlazada en promedio) las posiciona como la estructura de elección para aplicaciones que requieren operaciones de búsqueda frecuente.



4.4 Análisis de Árboles Rojo-Negro



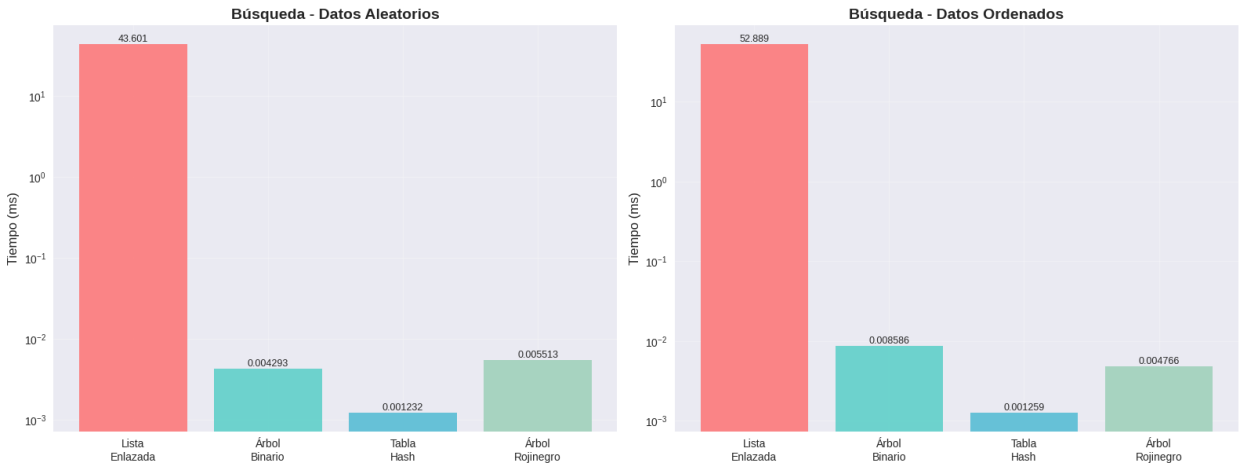
Los árboles rojo-negro muestran un comportamiento intermedio entre la estabilidad de las tablas hash y la sensibilidad de los árboles binarios simples. Los árboles rojo-negro mantienen propiedades de balanceo que garantizan complejidad O(log n) en el peor caso, lo cual se refleja en tiempos consistentes entre 0.004 y 0.006 ms para ambos patrones de datos.

Es interesante observar que, contrario a lo esperado, los datos aleatorios resultan ligeramente más favorables (13.5% más rápidos en búsqueda) que los datos ordenados. Esto sugiere que el overhead (costo adicional) del auto-balanceo puede ser mayor cuando se procesan secuencias ordenadas, debido a la necesidad de rotaciones más frecuentes para mantener las propiedades de balanceo del árbol.

4.5 Comparación de Operaciones de Búsqueda entre Estructuras

Los tiempos de búsqueda revelan grandes diferencias entre las estructuras de datos evaluadas. Las tablas hash mantienen su superioridad absoluta con tiempos prácticamente constantes (0.0012 ms promedio), siendo aproximadamente 35 veces más rápidas que las listas enlazadas (48.24 ms promedio). Los árboles rojo-negro demuestran consistencia con 0.0052 ms promedio, mientras que los árboles binarios simples muestran la mayor variabilidad según el patrón de entrada.

La diferencia más notable se observa en los árboles binarios simples, donde los datos ordenados duplican el tiempo de búsqueda respecto a los aleatorios (0.0086 vs 0.0043 ms). Esta sensibilidad contrasta la estabilidad de las tablas hash, que mantienen variaciones menores al 2% entre diferentes patrones de datos. Los árboles rojo-negro ocupan una posición intermedia, mostrando mayor estabilidad que los BST simples pero con ligeras variaciones (13.5% de diferencia) que sugieren overhead del auto-balanceo.

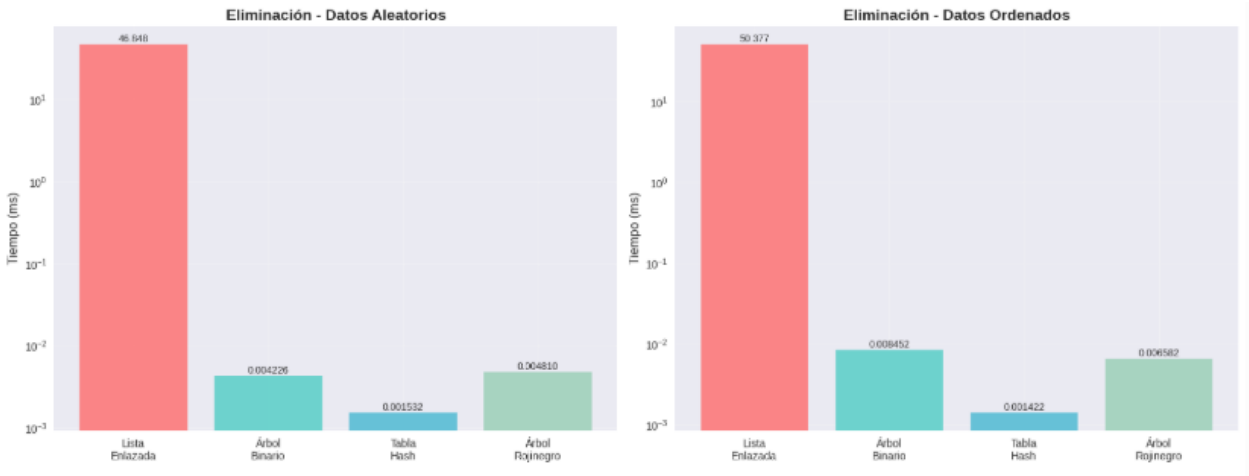


4.6 Comparación de Operaciones de Eliminación entre Estructuras

Las operaciones de eliminación siguen patrones similares a la búsqueda, pero con algunas diferencias. Las tablas hash mantienen su liderazgo con 0.0014 ms promedio, seguidas por los árboles binarios con datos aleatorios (0.0042 ms) y los árboles rojo-negro (0.0057 ms promedio). Las listas enlazadas nuevamente muestran los tiempos más elevados con 48.61 ms promedio.

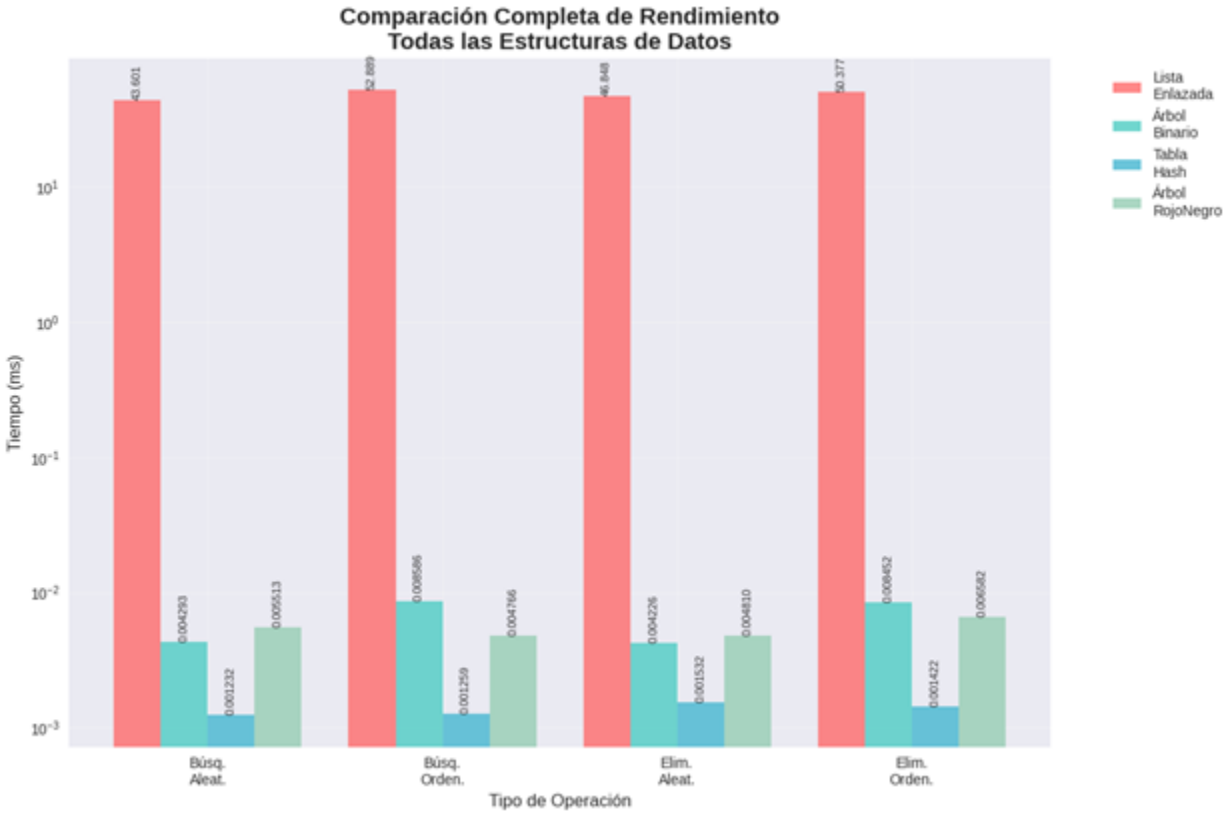
Un aspecto notable es que la eliminación en árboles binarios con datos ordenados (0.0085 ms) mantiene la misma tendencia de degradación que en búsqueda, confirmando el impacto de la

degeneración estructural. Los árboles rojo-negro exhiben un comportamiento peculiar donde los datos ordenados resultan ligeramente más favorables en eliminación (0.0048 vs 0.0066 ms), sugiriendo que las rotaciones necesarias para el rebalanceo pueden ser más eficientes durante eliminaciones secuenciales.



4.7 Implicaciones del Análisis Comparativo

Los resultados revelan que no existe una estructura de datos universalmente superior para todas las condiciones. Las tablas hash dominan en términos de velocidad absoluta y predictibilidad, pero los árboles balanceados como el rojo-negro ofrecen ventajas adicionales como ordenamiento implícito y eficiencia de memoria. Las estructuras simples como listas enlazadas y árboles binarios sin balanceo muestran limitaciones significativas que restringen su aplicabilidad a escenarios específicos. La sensibilidad del árbol binario simple al patrón de entrada muestra la importancia de considerar las características de los datos de entrada al seleccionar estructuras de datos. Esta observación respalda la recomendación de Cormen sobre la preferencia por estructuras auto-balanceadas en aplicaciones de propósito general.



5. Conclusiones

El objetivo de comparar los tiempos de ejecución de cuatro estructuras de datos y relacionar estos con las cotas teóricas se ha cumplido satisfactoriamente. Los experimentos realizados proporcionan evidencia real que valida las predicciones de complejidad asintótica y revela datos importantes en el comportamiento práctico de estas estructuras. Se confirma que las estructuras con complejidad teórica superior ofrecen ventajas significativas en términos de rendimiento absoluto. Las tablas hash, con su comportamiento prácticamente O(1), demuestran superioridad consistente independientemente del patrón de entrada. Los árboles rojo-negro validan su diseño como solución robusta que equilibra eficiencia y predictibilidad, manteniendo garantías de rendimiento O(log n) bajo todas las condiciones evaluadas.

El comportamiento del árbol binario simple ilustra dramáticamente los riesgos de utilizar estructuras no balanceadas en aplicaciones reales. Su degradación a comportamiento lineal con datos ordenados, termina resultando en mejores tiempos para ciertos casos específicos, demuestra la complejidad del análisis de rendimiento en condiciones reales versus predicciones teóricas.

Las listas enlazadas confirman su rol como estructura básica con limitaciones indiscutibles para operaciones de búsqueda intensiva, siendo superadas por alternativas más sofisticadas y en cuánto a su codificación, complejas. Su comportamiento predecible pero ineficiente las posiciona únicamente para casos donde la simplicidad de implementación compensa sus limitaciones de rendimiento. Considero que la lista enlazada simple, aunque no es una estructura de datos especialmente eficiente para aplicaciones reales, resulta útil como herramienta pedagógica para introducir conceptos fundamentales de estructuras de datos debido a su simplicidad.

Las implicaciones prácticas de este estudio sugieren que la selección de estructuras de datos debe basarse en un análisis conjunto de los patrones de datos esperados, la frecuencia de diferentes operaciones, y los requisitos de predictibilidad del rendimiento. Las tablas hash se proclaman como la opción preferida para aplicaciones donde la velocidad de acceso es crítica, mientras que los árboles balanceados ofrecen un compromiso apropiado cuando se requieren capacidades adicionales como ordenamiento eficiente.

Referencias

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3ª ed.). MIT Press.

Sedgewick, R., & Wayne, K. (2011). Algorithms (4ª ed.). Addison-Wesley Professional.

Knuth, D. E. (1998). The art of computer programming: Volume 3: Sorting and searching (2ª ed.). Addison-Wesley.

Robinson, S., Loshin, D., & Lewis, S. (2024, 2 de julio). Data structure. Search Data Management. https://www.techtarget.com/searchdatamanagement/definition/data-structure

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister. ACM Transactions on Modeling and Computer Simulation, *8*(1), 3–30. https://doi.org/10.1145/272991.272995

Intel Community. (2015, 7 de julio). Cache misses for sequential vs. random access patterns. https://community.intel.com/t5/Software-Tuning-Performance/Cache-misses-for-sequential-vs-random-access-patterns/td-p/1036280

Crucial.com. (2025). What does computer memory (RAM) do?

TechTarget. (s. f.). What is RAM (random access memory)?

Brainkart. (s. f.). Random access memory (RAM).

Vaia. (s. f.). What is the difference between sequential and random access?

Hacker News. (s. f.). Let this sink in. Random access into the cache has comparable performance to... https://news.ycombinator.com/item?id=22293045

University of California, Irvine. (2022). ICS 46 Spring 2022, Notes and Examples: Binary Search Trees. https://ics.uci.edu/~thornton/ics46/Notes/BinarySearchTrees/