Estructuras de Datos

Andres Murillo Murillo C15424

Resumen* — Las estructuras de datos son fundamentales para la optimización de procesos computacionales, afectando directamente la eficiencia de operaciones como inserción, búsqueda y eliminación. Este documento analiza y compara dos estructuras de datos esenciales: las listas enlazadas y los árboles binarios de búsqueda (BST). Las listas enlazadas (Linked Lists) permiten inserciones y eliminaciones eficientes, pero tienen limitaciones en operaciones de búsqueda debido a su naturaleza secuencial. Los árboles binarios de búsqueda (Binary Search Trees, BST), por otro lado, facilitan búsquedas más rápidas siempre que estén balanceados. Este estudio se centra en el análisis de la eficiencia teórica de estas estructuras y compara estas expectativas teóricas con los resultados observados mediante implementaciones en C++ y pruebas con datos aleatorios y secuenciales. Los hallazgos ofrecen una visión clara de las ventajas y limitaciones de cada estructura, ayudando en la selección adecuada para aplicaciones que requieren una comprensión profunda de su comportamiento teórico y práctico.

Palabras clave—Lista enlazada, árbol binario de búsqueda.

I. Introducción

En este estudio se realiza un análisis comparativo de dos estructuras de datos fundamentales en informática: las listas enlazadas (Linked Lists) y los árboles binarios de búsqueda (Binary Search Trees, BST). Estas estructuras son esenciales para la organización y manipulación eficiente de grandes cantidades de información, influyendo directamente en la eficiencia de operaciones como inserción, búsqueda y eliminación.

Las listas enlazadas permiten una gestión flexible de los elementos, facilitando inserciones y eliminaciones en cualquier posición sin la necesidad de reorganizar toda la estructura. Sin embargo, su eficiencia en operaciones de búsqueda es limitada debido a la naturaleza secuencial de su recorrido, presentando una complejidad de $O(n)$ en el peor de los casos. Esta característica las hace menos adecuadas para aplicaciones donde las búsquedas son frecuentes.

Por otro lado, los árboles binarios de búsqueda ofrecen una estructura jerárquica que facilita búsquedas, inserciones y eliminaciones más rápidas, siempre que el árbol esté equilibrado. La eficiencia de los BST se refleja en una complejidad promedio de $O(\log n)$ para estas operaciones, aunque en el peor de los casos, cuando el árbol está completamente desbalanceado, la complejidad puede degradarse a $O(n)$. Los BST son particularmente útiles en aplicaciones que requieren un acceso rápido y eficiente a los datos.

El propósito de este análisis es implementar estas estructuras en C++ y evaluar su eficiencia teórica en operaciones de inserción y búsqueda, utilizando datos aleatorios y secuenciales. Se recolectarán datos empíricos que se compararán con la eficiencia teórica de las estructuras, permitiendo así validar la teoría y entender mejor las ventajas y limitaciones de cada una en la práctica. Este enfoque proporciona una base sólida para la selección adecuada de estructuras de datos en aplicaciones que requieren una comprensión profunda de su comportamiento teórico y práctico, tales como bases de datos y sistemas en tiempo real.

Para una exploración más detallada de estos métodos y sus aplicaciones, se recomienda consultar "Introduction to Algorithms" de Thomas H. Cormen et al. (2022), una fuente ampliamente reconocida que ofrece una descripción exhaustiva de los principios y prácticas de las estructuras de datos.

II. Metodología

Para alcanzar los objetivos planteados en este estudio, se implementaron las estructuras de datos de listas enlazadas (Linked Lists) y árboles binarios de búsqueda (BST) utilizando C++ en un entorno Windows 11, ejecutándose en un procesador Intel i5 de 12ª generación. El código fuente, basado en el pseudocódigo de "Introduction to Algorithms" de Cormen et al. (2022), incluyó los métodos de inserción, búsqueda y eliminación para ambas estructuras. Se realizaron pruebas en dos escenarios: inserción de datos aleatorios y ordenados, utilizando conjuntos de datos con hasta 1,000,000 de elementos. La generación de números aleatorios se realizó con la biblioteca "random" de C++, y los tiempos de ejecución se midieron utilizando la biblioteca "chrono", promediando los resultados de al menos tres ejecuciones para garantizar precisión y consistencia. Los resultados se presentaron en gráficos y tablas comparativas, ofreciendo una visión clara de las diferencias en eficiencia entre listas enlazadas y árboles binarios de búsqueda, validando así la eficiencia teórica con datos empíricos y proporcionando una base sólida para la selección de estructuras de datos en aplicaciones que requieren un análisis detallado de su comportamiento teórico y práctico.

Cuadro I Tiempo de estructuras de datos: Lista doblemente enlazada y BST

	Tiempo (s) *Repetición*			
Prueba	1	2	3	Prom.
Ins. Aleatoria list	0.0549	0.0527	0.0568	0.0548
Ins. ordenada list	0.0415	0.0413	0.0400	0.0409
Busq. tras ins. Aleatoria list	26.9922	28.3276	28.8458	28.0552
Busq. tras ins. Ordenada list	29.3746	27.3991	27.7400	28.1712
Ins. aleatoria BST	0.6855	0.6108	0.6292	0.6418
Ins. ordenada BST	0.0376	0.0374	0.0430	0.0393
Busq. tras ins. aleatoria BST	0.0015	0.0013	0.0010	0.0012
Busq. tras ins. ordenada BST	34.8128	37.2887	35.2829	35.7948

La tabla muestra los tiempos de ejecución promedio en segundos para operaciones de inserción y búsqueda en listas enlazadas y árboles binarios de búsqueda (BST), con tres repeticiones por prueba. En las listas enlazadas, los tiempos de inserción son bajos, mientras que las búsquedas son significativamente más lentas debido a su naturaleza secuencial. En los BST, la inserción aleatoria es más lenta que la ordenada, ya que esta última degrada el árbol a una lista enlazada. Las búsquedas en BST son rápidas con inserción aleatoria pero mucho más lentas con inserción ordenada, reflejando el impacto del balanceo del árbol en el rendimiento.

III. RESULTADOS

Las listas doblemente enlazadas presentan una inserción eficiente debido a su estructura flexible, permitiendo agregar elementos en cualquier posición sin necesidad de reorganizar la estructura existente. En nuestras pruebas, la inserción aleatoria de elementos en una lista enlazada mostró tiempos de ejecución muy bajos, con un promedio de aproximadamente 0.0548 segundos. Este resultado es consistente con la teoría que establece que la inserción en una lista enlazada tiene una complejidad temporal de $\Theta(1)$ en el mejor caso, cuando se inserta al inicio de la lista, y $\Theta(n)$ en el peor caso cuando se inserta en el medio o al final de una lista de tamaño n. Para la inserción ordenada, los tiempos fueron aún menores, con un promedio de 0.0409 segundos. Esto se debe a que insertar elementos de manera secuencial evita el costo adicional de buscar la posición correcta para cada nuevo elemento, ya que simplemente se agregan al final de la lista. Este comportamiento también refleja la eficiencia esperada en la teoría de listas enlazadas.

La búsqueda en una lista enlazada tras la inserción de elementos aleatorios mostró un tiempo promedio de 28.0552 segundos para 10,000 búsquedas. Este resultado se alinea con la complejidad temporal de $\Theta(n)$ de la operación de búsqueda en listas doblemente enlazadas, ya que cada búsqueda puede requerir recorrer toda la lista en el peor de los casos. La variación en los tiempos de repetición puede atribuirse a factores de carga del sistema y a la naturaleza aleatoria de las claves buscadas. La búsqueda después de la inserción ordenada tuvo un tiempo promedio de 28.1712 segundos. Aunque ligeramente mayor que la búsqueda tras inserción aleatoria, esta diferencia no es significativa y puede ser atribuida a la misma complejidad $\Theta(n)$ inherente a la operación de búsqueda en listas enlazadas. La similitud en los tiempos de búsqueda para ambos tipos de inserción destaca que el método de inserción no afecta significativamente el tiempo de búsqueda en listas enlazadas.

Los resultados empíricos confirman la eficiencia teórica de las operaciones en listas enlazadas. La inserción es rápida y eficiente, especialmente en comparación con la búsqueda, que se ve limitada por la necesidad de recorrer secuencialmente los elementos. Estos resultados son consistentes con lo esperado para una estructura de datos que prioriza la flexibilidad en la gestión de elementos sobre la velocidad de acceso. Para aplicaciones donde la inserción y eliminación son más frecuentes que la búsqueda, las listas enlazadas son una elección adecuada. Sin embargo, para aplicaciones que requieren búsquedas rápidas, otras estructuras de datos como los árboles de búsqueda binaria pueden ser más apropiadas. En conclusión, las listas enlazadas muestran un rendimiento predecible y alineado con la teoría, validando su uso en contextos donde la inserción y eliminación dinámica de elementos son cruciales, pero donde la velocidad de búsqueda no es un factor crítico.

De acuerdo con el libro 'Introduction to Algorithms' de Cormen, Leiserson, Rivest y Stein, la búsqueda en una lista enlazada toma tiempo $\Theta(n)$ en el peor caso, ya que puede requerir buscar a través de toda la lista. La inserción al principio de la lista toma tiempo $\Theta(1)$, ya que solo se necesita ajustar unos pocos punteros. La inserción en cualquier posición de la lista toma tiempo $\Theta(1)$ si se tiene un puntero al nodo después del cual se va a insertar el nuevo nodo. La eliminación de un nodo de la lista toma tiempo $\Theta(1)$ si se tiene un puntero al nodo a eliminar, ya que solo se necesita ajustar unos pocos punteros. Las listas doblemente enlazadas permiten una gestión eficiente de la inserción y eliminación de elementos, con complejidades constantes $\Theta(1)$ para estas operaciones, mientras que la búsqueda de elementos sigue siendo lineal $\Theta(n)$. Estos resultados destacan la adecuación de las listas enlazadas para aplicaciones que requieren frecuentemente la inserción y eliminación de elementos, pero donde la velocidad de búsqueda no es un factor crítico.

Los árboles binarios de búsqueda (BST) son una estructura de datos que permite realizar operaciones eficientes de inserción, búsqueda y eliminación, siempre y cuando el árbol esté balanceado. En nuestras pruebas, se evaluaron diferentes escenarios para medir el rendimiento de estas operaciones.

Para la inserción aleatoria en BST, los tiempos promedios de ejecución fueron de 0.6418 segundos. Este tiempo refleja la complejidad *O(log n)* en el mejor caso, cuando el árbol permanece balanceado. Sin embargo, debido a la naturaleza aleatoria de la inserción, pueden ocurrir casos donde el árbol se desbalancee parcialmente, afectando ligeramente la eficiencia.

En el caso de la inserción ordenada, se optó por realizar inserciones siempre a la derecha para evitar comparaciones y mantener la operación constante. Esta estrategia degrada el árbol a una lista enlazada, lo que resulta en tiempos de inserción muy rápidos, con un promedio de 0.0393 segundos. Este resultado muestra una inserción eficiente en términos de tiempo, pero crea un árbol desbalanceado, afectando negativamente las operaciones de búsqueda.

La búsqueda tras la inserción aleatoria mostró un tiempo promedio de 0.0012 segundos, destacando la eficiencia de los BST cuando están balanceados o cuando la estructura resultante no está excesivamente desbalanceada. Este resultado es consistente con la complejidad *O(log n)* en el mejor caso, donde la búsqueda es rápida debido a la reducción exponencial del espacio de búsqueda en cada paso.

Sin embargo, la búsqueda tras la inserción ordenada mostró un tiempo promedio significativamente mayor, de 35.7948 segundos. Esta diferencia se debe a que el árbol se degrada a una lista enlazada cuando se insertan elementos de manera secuencial, resultando en una complejidad temporal de *O(n)* para la búsqueda. Este resultado pone de relieve la importancia de mantener el balance del árbol para optimizar tanto las inserciones como las búsquedas.

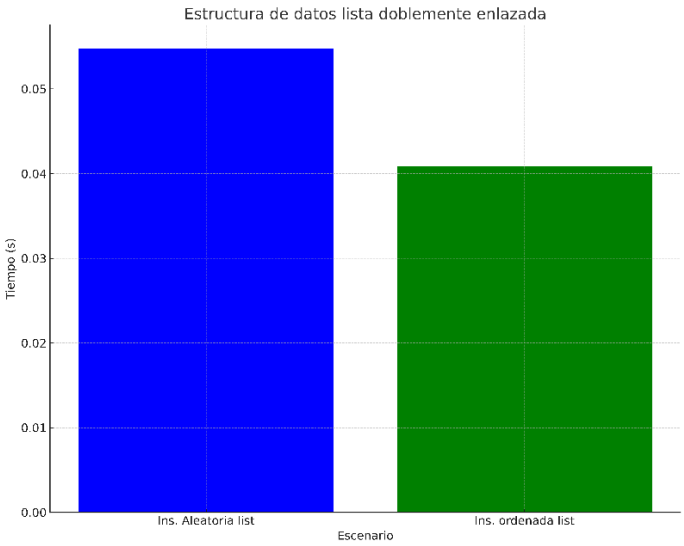
Los resultados empíricos obtenidos confirman la teoría sobre los BST. La inserción es rápida y eficiente en escenarios aleatorios, pero puede volverse subóptima en inserciones ordenadas debido a la degradación de la estructura. Las búsquedas son extremadamente rápidas en árboles balanceados, pero se vuelven ineficientes en árboles desbalanceados, como se observa en el caso de la inserción ordenada.

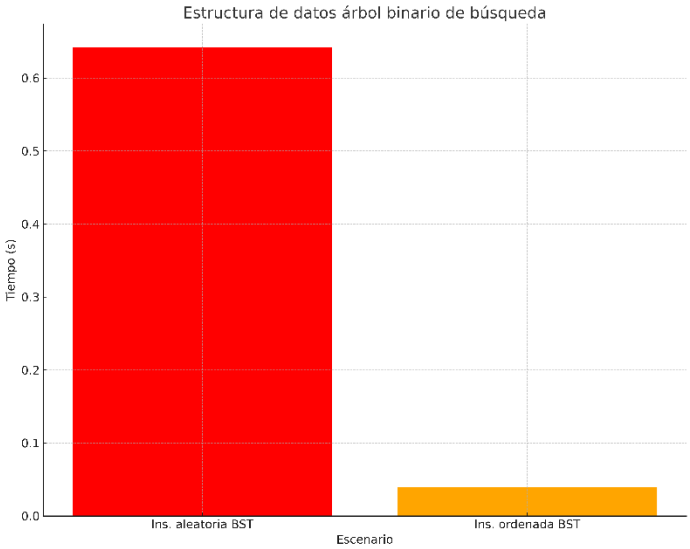
Para aplicaciones que requieren un acceso rápido y eficiente a los datos, los BST son una excelente opción, siempre y cuando se implementen mecanismos para mantener el balance del árbol, como en los árboles rojinegros o AVL. En contextos donde las inserciones ordenadas son comunes, es crucial considerar estructuras de datos alternativas o técnicas de balanceo para evitar la degradación de la eficiencia en las operaciones de búsqueda.

De acuerdo con el libro 'Introduction to Algorithms' de Cormen, et al. (2022), las operaciones básicas en un árbol binario de búsqueda tienen una complejidad proporcional a la altura del árbol. En el peor caso, las operaciones pueden tomar tiempo *O(n)* si el árbol es una cadena lineal de *n* nodos. Las operaciones de búsqueda, mínimo, máximo, sucesor y predecesor en un BST pueden ser implementadas para que cada una se ejecute en tiempo *O(h)*, donde *h* es la altura del árbol. La inserción y eliminación de nodos en un BST también tienen una complejidad de *O(h)* en el peor caso.

Las listas doblemente enlazadas y los árboles binarios de búsqueda (BST) son estructuras de datos fundamentales con características y comportamientos distintos. Las listas enlazadas destacan por su simplicidad y eficiencia en operaciones de inserción y eliminación, con tiempos de ejecución constantes en el mejor de los casos. Esta eficiencia se debe a que, para insertar o eliminar un nodo, solo es necesario ajustar unos pocos punteros, sin importar el tamaño de la lista. Sin embargo, la búsqueda en listas enlazadas puede ser ineficiente debido a su naturaleza secuencial. Dado que cada elemento de la lista debe ser examinado uno por uno, los tiempos de búsqueda son lineales en el peor caso, lo que puede ser una desventaja significativa cuando se manejan grandes volúmenes de datos.

Por otro lado, los BST ofrecen una estructura jerárquica que facilita búsquedas rápidas, siempre y cuando el árbol esté balanceado. En un árbol binario de búsqueda balanceado, la inserción y búsqueda pueden ser muy eficientes con una complejidad temporal logarítmica. Esto se debe a que, con cada comparación, la búsqueda puede descartar aproximadamente la mitad de los elementos restantes, reduciendo exponencialmente el espacio de búsqueda. Sin embargo, si el árbol se desbalancea, por ejemplo, debido a la inserción de elementos en orden ascendente o descendente, el BST puede degradarse a una estructura similar a una lista enlazada, resultando en una complejidad lineal para inserción y búsqueda. Para ilustrar mejor el comportamiento y las diferencias en rendimiento entre estas dos estructuras de datos, se presentan gráficos comparativos que muestran los tiempos promedio de inserción y búsqueda para escenarios de inserción aleatoria y ordenada. Estos gráficos proporcionan una visualización clara de cómo cada estructura maneja diferentes tipos de operaciones y tamaños de datos, permitiéndonos analizar si los tiempos de duración varían sustancialmente entre los escenarios de inserción de números aleatorios y secuenciales. Además, se indicarán las unidades de tiempo utilizadas en cada gráfico para facilitar la interpretación de los resultados.





Los gráficos presentados muestran los tiempos promedio de inserción para diferentes escenarios en las estructuras de datos de árboles binarios de búsqueda (BST) y listas doblemente enlazadas. En el gráfico del BST, se observa que la inserción aleatoria tiene un tiempo promedio significativamente mayor (0.6418 segundos) en comparación con la inserción ordenada (0.0393 segundos). Esta diferencia se debe a que la inserción ordenada se realizó siempre a la derecha, evitando comparaciones y manteniendo la operación constante, pero degradando el árbol a una lista enlazada, lo que resulta en una estructura desbalanceada. Esto contrasta con la inserción aleatoria, que mantiene la eficiencia logarítmica cuando el árbol está equilibrado.

En el gráfico de las listas doblemente enlazadas, los tiempos de inserción aleatoria y ordenada son relativamente similares, con promedios de 0.0548 y 0.0409 segundos, respectivamente. Esto refleja la eficiencia de las listas enlazadas para la inserción, ya que no requieren reordenar los elementos existentes independientemente de si los datos se insertan de forma aleatoria o secuencial. En comparación, las listas enlazadas muestran un rendimiento consistente en ambos escenarios de inserción, destacando su ventaja en la gestión dinámica de elementos sin comprometer la eficiencia en la inserción.

IV. Conclusiones

En conclusión, este estudio ha mostrado diferencias claras en el rendimiento de las listas doblemente enlazadas y los árboles binarios de búsqueda (BST). Las listas enlazadas se destacaron por su eficiencia en inserciones, con tiempos de ejecución bajos tanto en inserciones aleatorias como ordenadas. Sin embargo, las búsquedas en listas enlazadas son lineales y, por tanto, más lentas debido a su naturaleza secuencial. Por otro lado, los BST ofrecen búsquedas rápidas cuando están balanceados, pero su eficiencia puede degradarse significativamente si el árbol no está equilibrado. Las inserciones aleatorias en los BST son relativamente eficientes, mientras que las inserciones ordenadas, aunque rápidas, degradan el árbol a una lista enlazada, lo que resulta en un aumento considerable del tiempo de búsqueda. Estos resultados subrayan la importancia de elegir la estructura de datos adecuada según las necesidades específicas de la aplicación: las listas enlazadas son ideales para escenarios donde las inserciones y eliminaciones son frecuentes, debido a su flexibilidad y eficiencia. En contraste, los BST son más adecuados para aplicaciones que requieren búsquedas rápidas y frecuentes, siempre y cuando se implementen mecanismos para mantener el balance del árbol, como en los árboles rojinegros o AVL. En resumen, la selección entre listas enlazadas y BST debe basarse en el equilibrio necesario entre la eficiencia de inserción y búsqueda, optimizando el rendimiento según los requisitos específicos de cada aplicación.

V. Referencias

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L. y Stein, C. Introduction to Algorithms, IV ed. MIT Press, 2022.

[2] GeeksforGeeks. (2022, December 8). Insertion in Binary Search Tree (BST). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/insertion-in-binary-search-tree/

Andres Murillo Murillo

Estudiante de Ingeniería en Computacion en la Universidad de Costa Rica.

VI. APÉNDICE A

Código de los Algoritmos

Estructura de datos Lista doblemente enlazada Una lista doblemente enlazada es una estructura de datos donde cada nodo contiene un valor y dos punteros: uno al nodo siguiente y otro al nodo anterior. Esto permite recorrer la lista en ambas direcciones, facilitando la inserción y eliminación de nodos en cualquier posición de manera eficiente.

```
LIST-SEARCH(L, k)
1    x = L.head
2    while x ≠ NIL and x.key ≠ k
3        x = x.next
4    return x
```

```
LIST-PREPEND(L, x)
1    x.next = L.head
2    x.prev = NIL
3    if L.head ≠ NIL
4        L.head.prev = x
5    L.head = x
```

```
LIST-INSERT (x, y)
1    x.next = y.next
2    x.prev = y
3    if y.next ≠ NIL
4        y.next.prev = x
5    y.next = x
```

```
LIST-DELETE (L, x)
1    if x.prev ≠ NIL
2        x.prev.next = x.next
3    else L.head = x.next
4    if x.next ≠ NIL
5        x.next.prev = x.prev
```

```
LIST-DELETE (x)
1    x.prev.next = x.next
2    x.next.prev = x.prev
```

```
LIST-INSERT (x, y)
1    x.next = y.next
2    x.prev = y
3    y.next.prev = x
4    y.next = x
```

```
LIST-SEARCH (L, k)
1    L.nil.key = k
2    x = L.nil.next
3    while x.key ≠ k
4        x = x.next
5    if x == L.nil
6        return NIL
7    else return x
```

Estructura de datos Árbol binario de búsqueda Un árbol binario de búsqueda (BST) es una estructura de datos en la que cada nodo tiene un valor y hasta dos hijos: uno a la izquierda y otro a la derecha. Los nodos del subárbol izquierdo tienen valores menores que el nodo padre, y los del subárbol derecho tienen valores mayores. Esto permite realizar búsquedas, inserciones y eliminaciones de manera eficiente, generalmente en tiempo logarítmico cuando el árbol está balanceado.

```
INORDER-TREE-WALK (x)
1 if x ≠ NIL
2    INORDER-TREE-WALK (x.left)
3    print x.key
4    INORDER-TREE-WALK (x.right)
```

```
TREE-SEARCH (x, k)
1 if x == NIL or k == x.key
2    return x
3 if k < x.key
4    return TREE-SEARCH (x.left, k)
5 else return TREE-SEARCH (x.right, k)
```

```
ITERATIVE-TREE-SEARCH (x, k)
1 while x ≠ NIL and k ≠ x.key
2    if k < x.key
3        x = x.left
4    else x = x.right
5 return x
```

```
TREE-MINIMUM (x)
1 while x.left ≠ NIL
2    x = x.left
3 return x
```

```
TREE-MAXIMUM (x)
1 while x.right ≠ NIL
2    x = x.right
3 return x
```

```
TREE-SUCCESSOR (x)
1 if x.right ≠ NIL
2    return TREE-MINIMUM (x.right)
3 else
4    y = x.p
5    while y ≠ NIL and x == y.right
6        x = y
7        y = y.p
8    return y
```

```
TREE-INSERT (T, z)
1    x = T.root
2    y = NIL
3    while x ≠ NIL
4        y = x
5        if z.key < x.key
6            x = x.left
7        else x = x.right
8    z.p = y
9    if y == NIL
10       T.root = z
11   elseif z.key < y.key
12       y.left = z
13   else y.right = z
```

```
TRANSPLANT (T, u, v)
1    if u.p == NIL
2        T.root = v
3    elseif u == u.p.left
4        u.p.left = v
5    else u.p.right = v
6    if v ≠ NIL
7        v.p = u.p
```

```
TREE-DELETE (T, z)
1    if z.left == NIL
2        TRANSPLANT (T, z, z.right)
3    elseif z.right == NIL
4        TRANSPLANT (T, z, z.left)
5    else y = TREE-MINIMUM (z.right)
6        if y ≠ z.right
7            TRANSPLANT (T, y, y.right)
8            y.right = z.right
9            y.right.p = y
10       TRANSPLANT (T, z, y)
11       y.left = z.left
```

```
12        y.left.p = y
```