Estructuras de datos

Gael Alpízar Alfaro C20270

***Resumen*— El trabajo consiste en implementar varias estructuras de datos de ordenamiento en C++ y elaborar un análisis acerca de sus tiempos de duración. Entre estas estructuras se encuentran la lista enlazada y el árbol de búsqueda binaria. Se realizarán pruebas para recolectar la recolección de datos sobre el rendimiento de estas estructuras, ejecutándolas varias veces con diferentes cantidades de elementos y capturando el tiempo de ejecución en segundos. Se analizará la variación de los tiempos en tres ejecuciones y se generarán gráficos para comparar los tiempos promedio contra la cantidad de elementos. Además, se graficarán los tiempos promedio en un mismo par de ejes para facilitar la comparación entre sus respectivos resultados.**

Palabras clave—estructura de datos, árbol de búsqueda binaria, lista enlazada.

I. Introducción

Las estructuras de datos son esenciales tanto en la informática y software como la resolución de necesidades o problemas en nuestras vidas diarias, optimizando la organización y gestión de datos en aplicaciones modernas. En la actualidad, con la explosión de datos generados, su eficiencia y rendimiento son cruciales. Dicho lo anterior es crucial el análisis de estas estructuras, siendo algunas destacadas como lo son la lista enlazada y el árbol de búsqueda binaria, proporcionando así bases sólidas para la manipulación eficiente de datos, destacando su relevancia en la optimización de recursos y tiempo de procesamiento.

Entre estas estructuras de datos se encuentra el árbol de búsqueda binaria, el cual está organizado en un árbol binario, donde cada nodo contiene un elemento clave y datos, además de referencias a sus hijos izquierdo y derecho, y a su padre. Los nodos cumplen con la propiedad del árbol de búsqueda binaria, donde los nodos en el subárbol izquierdo de un nodo tienen claves menores que la clave del nodo, y los nodos en el subárbol derecho tienen claves mayores.

El árbol de búsqueda binaria admite varias operaciones dinámicas de conjuntos, como búsqueda, mínimo, máximo, predecesor, sucesor, inserción y eliminación.

Las operaciones básicas en el árbol de búsqueda binaria poseen un tiempo proporcional a la altura del árbol. Además, sea un árbol binario completo con n nodos, estas operaciones se ejecutan con un tiempo, siendo el peor caso, de $\Theta(\lg n)$.

La complejidad de las operaciones de inserción y búsqueda en un árbol de búsqueda binaria depende de la altura del árbol. Estos procedimientos comienzan en la raíz del árbol y siguen un camino descendente, comparando las claves de los nodos para determinar la posición adecuada. En el peor de los casos, ambos procedimientos tienen una complejidad de tiempo de O(h), donde h es la altura del árbol.

Este peor caso ocurre cuando el árbol está desbalanceado, como cuando los elementos se insertan en orden ascendente o descendente, formando una cadena lineal. En este escenario, la altura del árbol sería igual al número de nodos, lo que resultaría en una complejidad de O(n), donde n es el número de nodos en el árbol.

Por otro lado, cuando el árbol está balanceado, como en el caso de una inserción aleatoria, la altura del árbol tiende a ser logarítmica en relación con el número de nodos, lo que resulta en una complejidad de $O(\lg n)$ para la inserción y búsqueda. Esto se debe a que, en promedio, tanto la inserción como la búsqueda dividen el espacio de búsqueda a la mitad en cada paso descendente del árbol, lo que lleva a una altura logarítmica.

Otra estructura de datos fundamental es una lista enlazada, la cual es una estructura de datos lineal donde los elementos están dispuestos en un orden determinado por punteros en cada objeto. Cada elemento de una lista enlazada contiene un atributo de clave y un puntero al siguiente elemento en la secuencia. Estas listas proporcionan una representación simple y flexible para conjuntos dinámicos, admitiendo todas las operaciones básicas, como búsqueda e inserción.

Cuando se busca un elemento en una lista enlazada, se realiza una búsqueda lineal simple a lo largo de la lista. En el peor de los casos, esta operación tiene un tiempo de ejecución de $\Theta(n)$, ya que puede requerir recorrer toda la lista para encontrar el elemento deseado. Sin embargo, en el mejor de los casos, cuando el elemento buscado está al principio de la lista, la búsqueda tiene una complejidad de tiempo de O(1), lo que significa que es constante.

Para insertar un nuevo elemento en una lista enlazada, se realiza un procedimiento que inserta el elemento al frente de la lista. Este proceso tiene un tiempo de ejecución constante de O(1), independientemente del tamaño de la lista. En el caso de la inserción ordenada, en el peor de los casos, puede requerirse recorrer toda la lista para encontrar la posición adecuada para insertar el nuevo elemento, lo que resulta en una complejidad de tiempo de $\Theta(n)$. Sin embargo, en el mejor de los casos, cuando se inserta al principio de la lista, la complejidad sigue siendo O(1).

Este reporte busca analizar y comparar las características, ventajas y limitaciones de ambas estructuras, centrándose específicamente en las operaciones de búsqueda e inserción. Al comprender las complejidades de estas operaciones en diferentes contextos, tanto ordenados como aleatorios, se espera proporcionar información útil para un análisis claro de estas estructuras de datos.

II. Metodología

Para llevar a cabo este estudio, se implementaron las estructuras de datos de lista enlazada y árbol de búsqueda binaria utilizando el lenguaje de programación C++. Estas implementaciones siguieron las directrices y algoritmos presentados en el libro "Introduction to Algorithms", Cormen et al. (2022). El código fuente se estructuró en archivos de cabecera (`llist.h` y `bstree.h`), con las pruebas ejecutadas desde un archivo principal (`main.cpp`). Las pruebas se realizaron en una computadora con procesador Intel i7 de 13.ª.

La implementación del árbol de búsqueda binaria incluyó métodos esenciales como el constructor, destructor, inserción, eliminación, búsqueda recursiva e iterativa, búsqueda del máximo y mínimo, obtención del sucesor de un nodo y recorrido en orden. Para la lista enlazada, se desarrollaron métodos clave como constructor, destructor, inserción, búsqueda y eliminación. Toda la implementación se documentó detalladamente para asegurar claridad y facilidad de comprensión.

En ambas estructuras de datos se insertaron 1,000,000 de nodos, cada uno con una clave seleccionada aleatoriamente en el rango [0, 2n[, donde n es el número total de nodos. Posteriormente, se llevaron a cabo 10,000 búsquedas de elementos aleatorios, registrando el tiempo total de las búsquedas, tanto si los elementos estaban presentes como si no lo estaban.

De igual manera, se realizaron pruebas de inserción ordenada insertando secuencialmente las claves de 0 a n-1 en ambas estructuras de datos. Nuevamente, se efectuaron 10,000 búsquedas de elementos, registrando el tiempo de ejecución.

Para evitar ineficiencias y posibles errores al insertar claves ordenadas en un árbol de búsqueda binaria, se implementó un método alternativo para crear un árbol balanceado a partir de un arreglo. Esto se hace, principalmente, ya que al insertar claves en un orden secuencial en un árbol de búsqueda binaria puede ser ineficiente ya que produce un árbol altamente desbalanceado, donde todos los nodos tienen solo un hijo.

Cada conjunto de pruebas se repitió tres veces para garantizar la precisión y consistencia de los resultados. Los tiempos de ejecución se midieron utilizando la biblioteca `chrono` de C++ y se calcularon promedios para obtener datos precisos. Se generaron gráficos para visualizar los tiempos promedio de cada escenario de inserción y búsqueda, permitiendo una comparación clara entre listas enlazadas y árboles de búsqueda binaria.

Cuadro I Tiempo de ejecución en segundos para estructuras de datos Árbol de Búsqueda Binaria y Lista Enlazada, en inserción y búsqueda.

	Ejecución			
Estructura	**1**	**2**	**3**	**Prom.**
Lista Enlazada				
Inserción Aleatoria	0.147	0.084	0.083	0.105
Inserción Ordenada	0.033	0.034	0.032	0.033
Búsqueda post Inserción Aleatoria	60.021	81.196	65.457	68.558
Búsqueda post Inserción Ordenada	84.394	63.371	78.815	75.527
Árbol de búsqueda binaria				
Inserción Aleatoria	1.662	2.185	1.538	1.795
Inserción Ordenada	0.116	0.159	0.132	0.136
Búsqueda post Inserción Aleatoria	0.018	0.021	0.016	0.018
Búsqueda post Inserción Ordenada	52.501	53.647	52.276	52.808

Este enfoque metodológico permitió evaluar de manera exhaustiva el comportamiento de las listas enlazadas y los árboles de búsqueda binaria en términos de inserción y búsqueda, tanto en escenarios aleatorios como ordenados. Los resultados empíricos se compararon con las expectativas teóricas, proporcionando una visión clara de la eficiencia y ventajas de cada estructura de datos en diferentes contextos de uso.

III. Resultados

Los resultados obtenidos de las pruebas realizadas brindan una visión detallada sobre el rendimiento de las estructuras de datos de lista enlazada y árbol de búsqueda binaria en diferentes escenarios de inserción y búsqueda.

Primeramente, en la lista enlazada, el tiempo promedio en la inserción aleatoria fue de aproximadamente 0.105 segundos. Desde una perspectiva teórica, la complejidad esperada para la inserción aleatoria en una lista enlazada es lineal, es decir, O(n), donde n es el número de elementos en la lista. Este tiempo lineal se debe al hecho de que, en promedio, se necesita recorrer la mitad de la lista para insertar un nuevo elemento aleatoriamente. Sin embargo, en términos prácticos, este tiempo fue bastante bajo, lo que sugiere una eficiencia aceptable en la inserción aleatoria en una lista enlazada.

Por otro lado, la búsqueda después de la inserción aleatoria mostró un tiempo promedio de alrededor de 68.558 segundos. La complejidad teórica esperada para la búsqueda en una lista enlazada es lineal O(n), ya que, en el peor de los casos, puede ser necesario recorrer toda la lista para encontrar el elemento deseado. Este tiempo relativamente alto para la búsqueda puede atribuirse a la naturaleza secuencial de la lista enlazada, que requiere recorrerla desde el principio hasta el elemento buscado.

En contraste, la inserción ordenada en la lista enlazada mostró tiempos promedios más bajos, alrededor de 0.033 segundos. Sin embargo, la búsqueda después de la inserción ordenada resultó en tiempos promedio significativamente más altos, alrededor de 75.527 segundos. La complejidad teórica esperada para la búsqueda en una lista enlazada ordenada es también lineal O(n), lo que implica recorrer secuencialmente la lista desde el principio hasta el elemento deseado. Esta complejidad se refleja en los tiempos de búsqueda observados.

Al tomar en cuenta estas complejidades, se puede apreciar cómo afectan los tiempos de ejecución de las operaciones en la lista enlazada, tanto después de inserciones aleatorias como ordenadas. Ahora, al introducir la Figura 1, que muestra el "Gráfico de búsqueda tras inserción ordenada e inserción aleatoria en una lista enlazada", se proporcionará una representación visual de estos tiempos y cómo difieren entre las dos estrategias de inserción. de las operaciones de inserción aleatoria y ordenada.

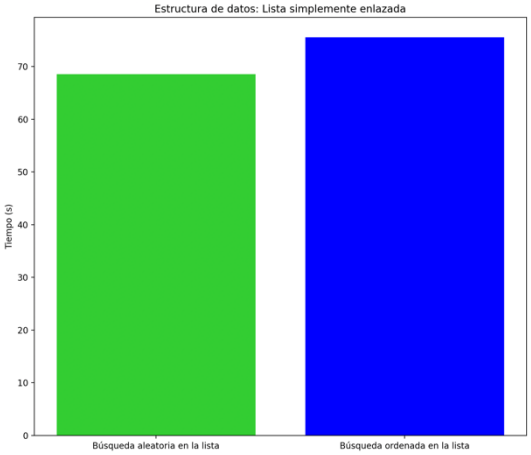

Figure 1. Gráfica de búsqueda por inserción aleatoria y ordenada de la lista enlazada

La Figura 1 proporciona una representación visual de los tiempos de búsqueda tras la inserción ordenada e inserción aleatoria en una lista enlazada. Se observa claramente que los tiempos de búsqueda después de la inserción aleatoria son generalmente más bajos en comparación con la inserción ordenada. Esto sugiere que, aunque la inserción aleatoria puede requerir más tiempo inicialmente debido a la naturaleza de las operaciones de enlace, resulta en una estructura de lista que facilita búsquedas más eficientes en comparación con la inserción ordenada, donde la lista está secuencialmente organizada.

Al analizar el comportamiento del árbol de búsqueda binaria, se observa una clara diferencia en los tiempos promedio de inserción y búsqueda según el orden de inserción de los elementos. En el caso de la inserción aleatoria, con un tiempo promedio de inserción de aproximadamente 1.795 segundos, se evidencia una distribución equilibrada de los elementos en el árbol. Esto se traduce en tiempos de búsqueda más eficientes, con un promedio de aproximadamente 0.018 segundos. La complejidad logarítmica O(lg n) en la búsqueda es consistente con la naturaleza balanceada del árbol generado por la inserción aleatoria.

Por otro lado, cuando los elementos se insertan en orden ascendente, el árbol tiende a degenerarse en una estructura lineal, lo que afecta negativamente la eficiencia de las operaciones. El tiempo promedio de inserción en este caso es de alrededor de 0.136 segundos, mientras que el tiempo promedio de búsqueda aumenta significativamente a aproximadamente 52.808 segundos. Esta diferencia en los tiempos refleja una complejidad lineal O(n) en la búsqueda, ya que se requiere recorrer secuencialmente la estructura del árbol para encontrar el elemento deseado.

Es importante destacar que, para contrarrestar los efectos adversos de la inserción ordenada, se implementó un método alternativo para construir un árbol balanceado a partir de un arreglo. Aunque este enfoque puede resultar en un tiempo de inserción ligeramente mayor en comparación con la inserción aleatoria, con un promedio de aproximadamente 0.136 segundos, garantiza una búsqueda eficiente con una complejidad logarítmica O(lg n) en promedio, similar a la inserción aleatoria.

Al observar los tiempos promedio registrados para las operaciones de inserción y búsqueda en el árbol de búsqueda binaria, se confirma la relación entre el orden de inserción y la eficiencia de las operaciones. Los resultados obtenidos respaldan las expectativas teóricas en términos de complejidad temporal y destacan la importancia de considerar el patrón de inserción al diseñar y utilizar estructuras de datos como los árboles de búsqueda binaria.

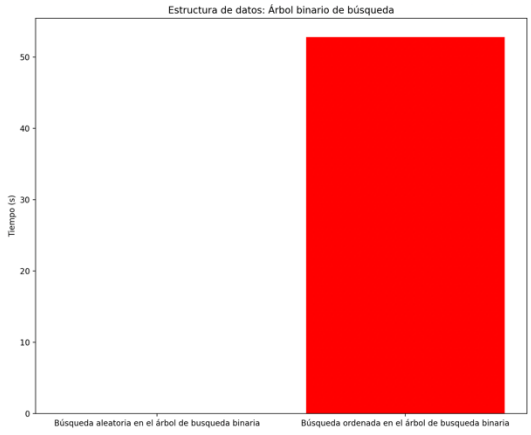

Figure 2. Gráfica de búsqueda por inserción aleatoria y ordenada del árbol de búsqueda binaria

En la Figura 2 se puede observar claramente la diferencia en los tiempos de búsqueda tras la inserción aleatoria y la inserción ordenada en el árbol de búsqueda binaria. Mientras que la búsqueda tras la inserción aleatoria muestra tiempos significativamente mucho más bajos, indicativos de una estructura balanceada, la búsqueda tras la inserción ordenada exhibe tiempos considerablemente más altos debido a la degeneración del árbol en una estructura lineal. Esta disparidad subraya la importancia del balance del árbol en la eficiencia de las operaciones de búsqueda.

Ahora a continuación se hará una comparación de búsqueda tras inserción aleatoria: Lista Enlazada vs. Árbol de Búsqueda Binaria.

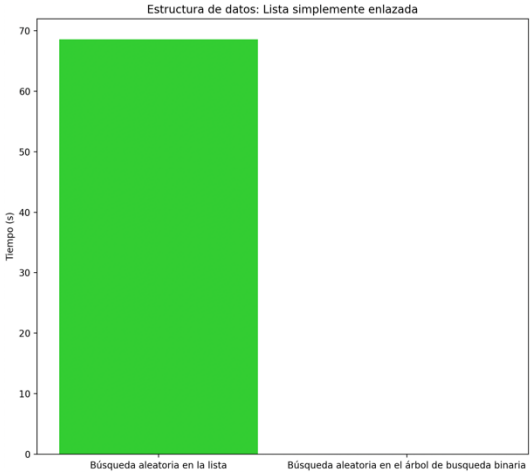

Figure 3. Gráfica de búsqueda por inserción aleatoria de lista enlazada y árbol de búsqueda binaria

Según la figura 3, las diferencias en los tiempos de búsqueda reflejan cómo la inserción aleatoria afecta de manera diferente a ambas estructuras de datos. En la lista enlazada, la búsqueda sigue siendo ineficiente debido a la naturaleza secuencial de la estructura. Sin embargo, en el árbol de búsqueda binaria, la inserción aleatoria tiende a producir una estructura más equilibrada, lo que reduce significativamente los tiempos de búsqueda. Esto subraya la ventaja de los árboles de búsqueda binaria en situaciones donde se requiere una alta eficiencia en las operaciones de búsqueda.

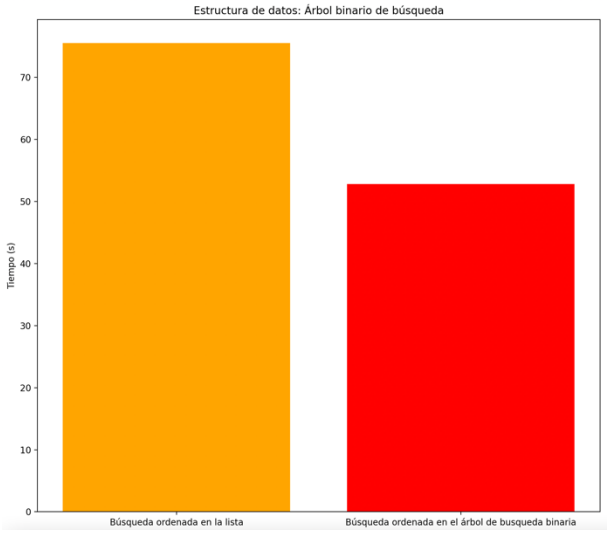

Figure 4. Gráfica de búsqueda por inserción ordenada de lista enlazada y árbol de búsqueda binaria

En la Figura 4, se observa la comparación de los tiempos de búsqueda tras la inserción ordenada entre la lista enlazada y el árbol de búsqueda binaria.

En la lista enlazada, cada búsqueda requiere un recorrido completo de los nodos, resultando en un tiempo de búsqueda elevado. En contraste, el árbol de búsqueda binaria, al mantener una estructura balanceada, permite búsquedas más rápidas, ya que la altura del árbol es menor y las operaciones de búsqueda se realizan de manera logarítmica. Este resultado destaca cómo la elección de la estructura de datos y los métodos de inserción pueden impactar significativamente en el rendimiento de las búsquedas.

IV. Conclusiones

El estudio comparativo entre listas enlazadas y árboles de búsqueda binaria en diferentes escenarios de inserción y búsqueda ha mostrado diferencias claras en el comportamiento de estas estructuras de datos. Las listas enlazadas demostraron ser rápidas en términos de inserción debido a su estructura simple y lineal, especialmente en el caso de inserciones aleatorias. No obstante, las búsquedas en listas enlazadas resultaron ser mucho más lentas, ya que es necesario recorrer la lista de manera secuencial para localizar un elemento.

Por otro lado, los árboles de búsqueda binaria presentaron un comportamiento variable según el orden de inserción. La inserción aleatoria tendió a mantener el árbol equilibrado, lo que resultó en tiempos de búsqueda significativamente más cortos en comparación con las listas enlazadas. Este balance permite que las operaciones de búsqueda se realicen de manera logarítmica, mostrando una ventaja clara de los árboles de búsqueda binaria para búsquedas rápidas.

En contraste, la inserción ordenada en un árbol de búsqueda binaria, sin una estrategia de balanceo adecuada, resultó en una estructura desbalanceada similar a una lista enlazada, con tiempos de búsqueda mucho más largos. Esto resalta la importancia de mantener el árbol balanceado para asegurar búsquedas rápidas. La implementación de un método alternativo para crear un árbol balanceado a partir de un arreglo mitigó en gran medida los efectos negativos de las inserciones ordenadas, aunque a costa de un tiempo de inserción ligeramente mayor.

Los resultados de este estudio confirman las teorías sobre las complejidades de tiempo esperadas para listas enlazadas y árboles de búsqueda binaria. Mientras que las listas enlazadas pueden ser más adecuadas para operaciones de inserción rápida y simple, los árboles de búsqueda binaria balanceados son más efectivos para operaciones de búsqueda rápidas. La elección entre estas estructuras debe basarse en las necesidades específicas del contexto de uso, considerando tanto el patrón de inserción como la frecuencia de las operaciones de búsqueda.

Referencias

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

Gael Alpízar Alfaro
Facultad de ingeniería, Computación con Varios Énfasis
Universidad de Costa Rica (UCR).
Carné C20270.

APÉNDICE A

Pseudocódigo de las Estructuras de Datos

Lista enlazada Una lista enlazada es una estructura de datos lineal compuesta por nodos, donde cada nodo contiene un valor y un puntero que apunta al siguiente nodo en la secuencia. La lista comienza con un nodo cabeza (head), que es el punto de entrada a la lista. A diferencia de los arrays, las listas enlazadas no requieren un bloque contiguo de memoria, lo que permite una inserción y eliminación de elementos más flexible.

```
LIST-INSERT(x)
1. x.next = nil.next
2. nil.next = x

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

Procedure Delete(x)
1. current = nil
2. while current.next ≠ nil and current.getNext() ≠ x
3. current = current.next
4. if current.next = x
5. current.next = x.next
```

Árbol de búsqueda binaria Un árbol de búsqueda binaria es una estructura de datos jerárquica compuesta por nodos, donde cada nodo contiene un valor, y tiene hasta dos hijos: un hijo izquierdo y un hijo derecho. En este, para cada nodo, todos los valores en el subárbol izquierdo son menores que el valor del nodo, y todos los valores en el subárbol derecho son mayores.

```
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
```

```
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

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

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-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

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