

# Reporte de resultados: Tarea Programada 2

Jeremy Rojas, B96804 <jeremy.rojascerna@ucr.ac.cr>

**Resumen**—En este trabajo se mide una serie de métodos en cuatro estructuras de datos para comparar sus tiempos de ejecución. Para esto, cada estructura de datos se implementó en C++ y se midió los tiempos de duración de los métodos. Los resultado fueron tabulados y graficados para facilitar el análisis. Se concluye que cada estructura de datos tiene un nicho, por lo que no hay estructura de datos que funcione para todo problema.

**Abstract**—This work measures a series of methods across four data structures to compare their execution times. For this purpose, each data structure was implemented in C++ and the duration times of the methods were measured. The results were tabulated and graphed to facilitate analysis. It is concluded that each data structure has its niche, so there is no single data structure that works for every problem.

**Palabras clave**—inserción, búsqueda, eliminación, estructura de datos, lista simplemente enlazada, árbol binario, árbol rojinegro, tabla de dispersión.

**Keywords**—insertion, search, deletion, data structure, singly linked list, binary tree, red-black tree, hash table.

## I. INTRODUCCIÓN

En este trabajo se mide y se compara los tiempos de ejecución de tres procedimientos en cuatro estructuras de datos estudiadas en el curso CI-0116 Análisis de Algoritmos y Estructuras de Datos. Estas estructuras de datos son las siguientes:

- Lista Simplemente Enlazada (Singly Linked List).
- Árbol Binario (Binary Search Tree).
- Árbol Rojinegro (Red-Black Tree).
- Tabla de Dispersión (Chained Hash Table).

De esta manera, se busca comparar los tiempos de ejecución entre cada estructura de datos para corroborar, por un lado, los límites asintóticos denotados en Cormen et al. [1], y por otro, determinar qué estructura de dato sería la ideal según las necesidades del programa.

## II. METODOLOGÍA

Para lograr lo propuesto se implementó cada estructura de datos en C++. El programa de prueba genera dos arreglos de un millón de enteros positivos: en el primero, los números están en orden aleatorio, y en el segundo, están ordenados de menor a mayor. Cada arreglo es insertado en la estructura de datos, en instancias distintas de la estructura, y luego cada instancia busca diez mil números aleatorios. Finalmente, cada estructura elimina diez mil valores.

La generación de números aleatorios se realizó con los objetos *generator* y *distribution* de la biblioteca estándar

<random>. La medición del tiempo se realizó con con el objeto *high\_resolution\_clock* de la biblioteca estándar <chrono>. La funcionalidad de éste último está encapsulada en un objeto *Timer* para facilitar el llamado de los métodos de temporizador.

Cuando el programa inicia, y se le ordena que pruebe una estructura de datos, éste genera un primer arreglo de números, en orden aleatorio, cuyos valores se insertan en la estructura de datos. Ya con estos datos almacenados, se realiza un proceso de búsqueda de diez mil valores en un intervalo de  $[0, 3n[$ , donde  $n$  es el tamaño del arreglo (en estas pruebas, un millón). Después de estas busquedas, se eliminan diez mil valores en ese mismo intervalo. Cada proceso (insertar, buscar, eliminar) se cronometra con un *Timer*. El mismo procedimiento se realiza inmediatamente después con un arreglo que contiene valores ordenados de menor a mayor. Por estructura de datos, todo esto se realiza tres veces.

## III. RESULTADOS

Cuadro I Tiempo de ejecución de los procedimientos

Estructura	Orden	Tiempo promedio(ms)		
		Método		
SLL	A	38.8875	24545.4333	28765.7333
	O	10.8315	2541.3667	31440.5333
		24.8595	24979.9	30103.1333
BST	A	1341.4367	54.7013	59.7859
	O	133.1283	4.4456	4.99
		737.2825	29.5734	32.3879
RBT	A	775.3283	7.8534	9.0630
	O	323.4377	5.0610	7.2027
		549.3830	6.4572	8.1328
CHT	A	366.1753	4.883	5.0147
	O	28.4395	5.3477	5.1991
		197.3074	5.1154	5.1069

Los tiempos promedio de ejecución de los tres procedimientos (métodos) se muestran en el Cuadro I. En la columna de *Orden* lo que se denota es el orden de los valores en el arreglo cuando éstos son insertados a la estructura de datos. *A* denota que el arreglo estaba en orden aleatorio; *O* denota que el arreglo estaba ordenado de menor a mayor. Los valores debajo de las filas *A* y *O* son el promedio entre estos dos.

La figura 1 muestra los tiempos de las búsquedas de valores insertados en orden aleatorio. La figura 2 muestra los tiempos de las búsquedas de valores insertados en orden, de menor a mayor. La figura 3 muestra los tiempos de eliminación de los valores insertados en orden aleatorio. Finalmente, la figura 4 muestra los tiempos de eliminación de los valores insertados en orden, de menor a mayor. Todos los cuatro gráficos están en escala logarítmica.

Figura 1 Tiempos promedio de búsqueda de valores insertados en orden aleatorio.

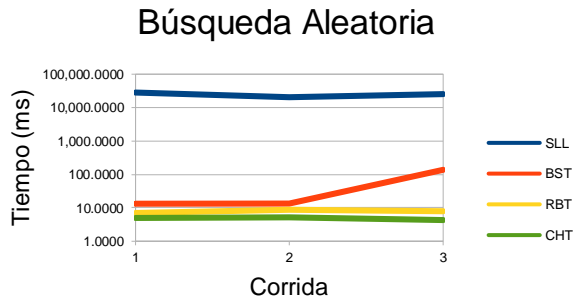


Figura 2 Tiempos promedio de búsqueda de valores insertados en orden.

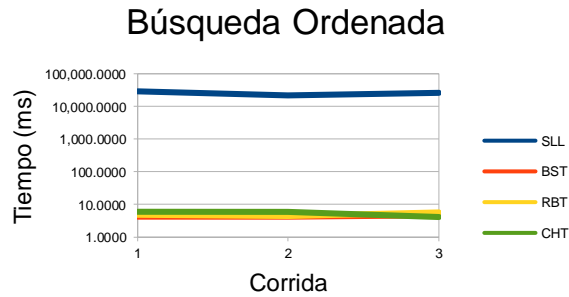


Figura 3 Tiempos promedio de eliminación de valores insertados en orden aleatorio.

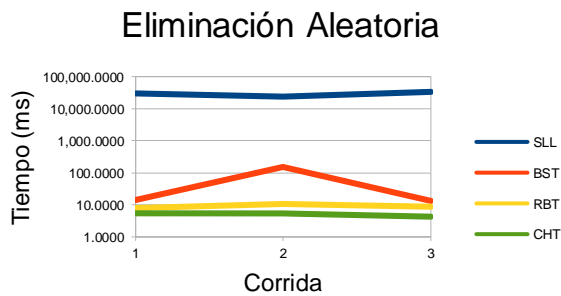
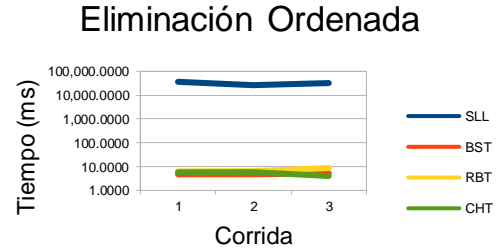


Figura 4 Tiempos promedio de eliminación de valores insertados en orden.



#### IV. DISCUSIÓN

La siguiente tabla recopila las cotas descritas para los métodos de búsqueda y eliminación en Cormen et al. [1], en los capítulos 10, 11, 12 y 13. Conviene resaltar que estas cotas describen el caso promedio y el peor caso, en ese orden. En caso de que haya una única cota, esta se toma como igual para ambos casos.

*Cuadro II Cotas de los métodos de búsqueda y eliminación según estructura de datos*

Estructura de datos	Búsqueda	Eliminación
SLL	$\Theta(k) / \Theta(n)$	$\Theta(k) / \Theta(n)$
BST	$O(h)$	$O(h)$
RBT	$\Theta(\lg n)$	$\Theta(\lg n)$
CHT	$\Theta(1+\alpha)$	$\Theta(1+\alpha)$

En estas cotas,  $k$  es la llave de búsqueda,  $n$  es el tamaño de la entrada,  $h$  es la altura del árbol, y  $\alpha$  es el factor de carga (que en el caso de las pruebas era 1). A la luz de estas cotas, los gráficos se corresponden bastante bien con la teoría.

Primero, en la búsqueda aleatoria tenemos que las estructuras están en este orden, de la cota más grande a la más pequeña: SLL, BST, RBT, CHT. Este mismo orden es el que presentan las cotas, en el peor de los casos, para cada una de estas estructuras de datos. No solo eso, sino que además ese mismo orden se ve reflejado bastante bien en la eliminación aleatoria, y en las cotas que corresponden al peor de los casos.

Segundo, en la búsqueda y eliminación ordenadas tenemos que las Listas Simplemente Enlazadas siguen siendo las más lentas. Sin embargo, en el caso de los otros tres algoritmos, tenemos tiempos sumamente cercanos, particularmente entre los Árboles Rojinegros y las Tablas de Dispersión. Esto habla de la virtud de estas estructuras de datos en balancear la carga de distribución, lo que acelera el proceso de búsqueda

posteriormente. Sí, la inserción es más lenta, pero es el precio de colocar los datos en un lugar de rápido acceso. Los Árboles Binarios normales no se quedan muy atrás: hay que recordar que un Árbol Binario perfectamente balanceado tiene una altura ( $\lg n$ ), de manera que las cotas de un Árbol Binario se vuelven  $O(\lg n)$ ; prácticamente un Árbol Rojinegro.

A pesar de estos resultados, hay que notar en el Cuadro I que los tiempos de inserción para las Listas Simplemente Enlazadas son los más bajos, de lejos. Además, los tiempos de inserción para los Árboles Rojinegros son, en promedio, mejores que los de los Árboles Binarios, lo que tiene sentido dado que los primeros son derivados de los segundos; el costo de realizar los ajustes para respetar las propiedades rojinegras se ve recompensado en tiempo menores.

Finalmente, las Tablas de Dispersión no son las más lentas, pero tampoco las más rápidas. Como siempre, éstas dependen de la función de dispersión y cómo ésta interactúa con los valores de entrada, las llaves, y sus posiciones. En el caso de la inserción de valores ordenados, las Tablas de Dispersión compiten con las Listas Simplemente Enlazadas, lo que apunta a un número bajo de colisiones y, por tanto, de sondeos. Sin embargo, es importante recalcar que las Tablas de Dispersión implementadas están encadenadas con Listas Doblemente Enlazadas, por lo que los tiempos adicionales durante la inserción aleatoria resultan sorprendentes.

Como último *addendum*, la paridad entre tiempos de búsqueda y tiempos de eliminación está respaldada por el que,

para la eliminación de un dato, es necesario su búsqueda primero.

## V. CONCLUSIONES

A partir de los resultados se puede concluir que, si se sabe de antemano que los valores de entrada van a ser aleatorios, el costo de implementar un Árbol Rojinegro o una Tabla de Dispersión se ve compensado con las velocidades de búsqueda y, por extensión, de eliminación. En cambio, si se sabe que los datos vienen completamente ordenados, un simple Árbol Binario ahorra costos de programación sin ser particularmente lento. Las Listas Simplemente Enlazadas tienen un nicho en casos donde lo que se requiera es una inserción y extracción rápida de datos (ésto último no se probó en este trabajo, pero está respaldado por el que las Listas Simplemente Enlazadas suelen ser el esqueleto de las Colas y las Pilas).

## REFERENCIAS

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. Cambridge, Massachusetts: The Mit Press, 2022.