Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Informe Tarea 5

Comparación de Árboles Balanceados

Fecha: 17/04/2016

Autor: Marcial Díaz Flores

Email: marcial.diaz@ing.uchile.cl

Código del curso: CC3001-2

Introducción:

Un árbol binario de búsqueda (ABB), también llamado BST (Binary Search Tree), es un tipo de árbol binario (árbol cuyos nodos poseen a lo más dos nodos hijos). Este tipo de árbol binario se caracteriza por cumplir con las siguientes condiciones: el subárbol izquierdo de cualquier nodo (si es que no está vacío) posee valores menores al valor que contiene dicho nodo, y el subárbol derecho (mientras no sea vacío) posee valores mayores.

Un árbol AVL es un tipo particular de árbol binario de búsqueda, siendo AVL un acrónimo obtenido del nombre de los matemáticos rusos Adelson-Velskii y Landis. Éste árbol fue el primer árbol binario de búsqueda auto-balanceable, dicho balanceo se consigue haciendo rotaciones en los nodos luego de cada inserción y borrado. Con el balanceo se logra mantener el árbol equilibrado, esto garantiza una mejor eficiencia de búsqueda con respecto al ABB normal.

El árbol Rojo-Negro es otro árbol binario de búsqueda equilibrado. Su estructura original fue creada en 1972 por el científico alemán Rudolf Bayer. Los nodos de este árbol poseen un atributo de color, el cual puede ser rojo o negro. Gracias a rotaciones y cambios de color tras cada inserción y borrado, el árbol se mantiene balanceado. A pesar de ser complejo tiene un buen tiempo de ejecución para sus operaciones en el peor caso y es eficiente en la práctica.

En esta tarea se implementaron los tres tipos de árbol binario mencionados, y se comparó el rendimiento de dichos árboles en diversos escenarios.

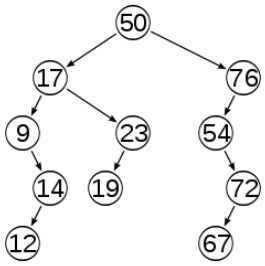
Análisis del Problema:

El problema principal es implementar la interfaz ArbolBinario a través de un ABB, un AVL y un Árbol Rojo-Negro; los cuales en consecuencia deben poseer los métodos de inserción y búsqueda. Para esto se supuso que los datos que se insertan o buscan en dichos árboles son mayores o iguales a 0, y que en el caso del AVL y del Árbol Rojo-Negro nunca se insertan valores que ya están en el árbol.

Para implementar la clase ABB (Árbol Binario de Búsqueda normal) se usaron los algoritmos estándar de inserción y búsqueda en un ABB normal. Para la inserción se compara el dato a insertar con el de la raíz del árbol. Si el dato a insertar es más pequeño se llama recursivamente con el subárbol izquierdo, en caso contrario con el derecho, hasta encontrar el lugar de la inserción. Para la búsqueda se compara el dato a buscar con el de la raíz del árbol. Si el dato a buscar es más pequeño que el de la raíz, se busca en el subárbol izquierdo de la raíz, en caso contrario en el subárbol derecho. Si ambos datos son iguales el dato pertenece al árbol.

Su complejidad promedio de búsqueda e inserción es O(log n), pero en el peor caso (los nodos tienen a lo más un hijo) se comporta como lista enlazada, por lo que la complejidad de búsqueda e inserción en el peor caso es O(n).

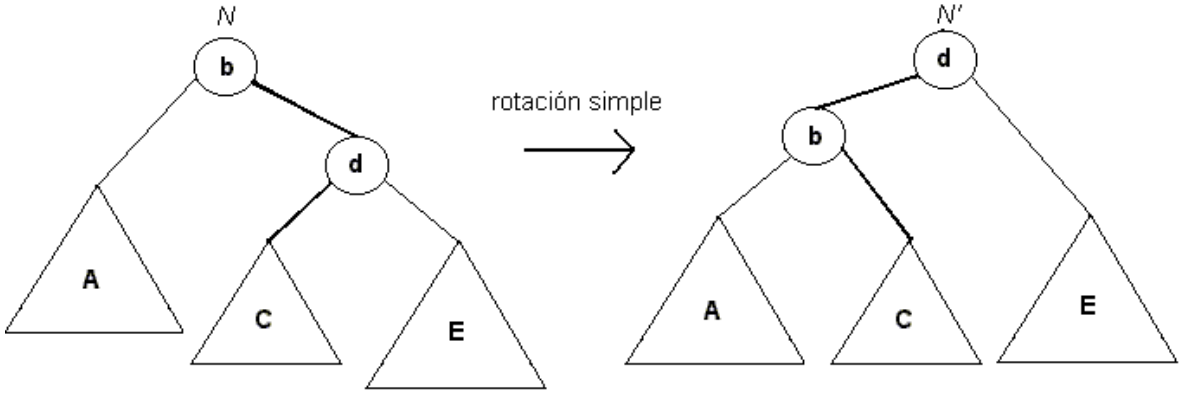
ABB normal (desbalanceado):



En el caso del AVL, los algoritmos de inserción y búsqueda son casi idénticos a los de un ABB normal, a excepción de la inserción, en la cual se debe asegurar que el árbol cumple con la condición de equilibrio. Dicha condición consiste en que la altura de los subárboles izquierdo y derecho de un nodo pueden diferir a lo más en 1. Si esta condición debe cumplirse para todos los nodos del árbol, en caso contrario no es un AVL válido.

Esta condición de equilibrio se puede ver alterada tras cada inserción, para solucionar este problema se efectúan rotaciones en los nodos tras alterar la condición de equilibrio. Estas rotaciones consisten en cambiar las referencias de los nodos para cambiar la altura del árbol sin que deje de ser un ABB.

Rotación Simple:



Esta condición de equilibrio garantiza que la altura es del orden log(n), con n la cantidad de elementos del AVL. Además sus operaciones son de complejidad logarítmica O(log n).

En el caso del árbol Rojo-Negro el balanceo se asegura mediante rotaciones de los nodos y cambios de color. El balanceo de este árbol produce una regla importante, el camino más largo desde la raíz hasta una hoja no es más largo que dos veces el camino más corto desde la raíz a una hoja, y gracias a esto el árbol se encuentra equilibrado. Por lo tanto sus operaciones también son de complejidad logarítmica O(log n). Además este árbol posee un peor caso eficiente de complejidad logarítmica.

Solución del Problema:

Para implementar la interfaz ArbolBinario se implementaron tres clases con los tres tipos de árboles: ABB (normal), AVL y Árbol Rojo-Negro.

Clase ABB:

La clase ABB posee una clase auxiliar llamada NodoABB, que representa a los nodos del árbol, con un dato (entero) y referencias al nodo izquierdo y al derecho. La clase ABB posee como campo una instancia de NodoABB que representa a la raíz del árbol.

El método insertarNodo(int x, NodoABB nodo) recibe como parámetros un entero y un nodo, luego crea un nodo nuevo a partir del entero y lo inserta al nodo parámetro. Si el nodo parámetro es nulo, se retorna el nuevo nodo, si el dato parámetro es menor al del nodo parámetro se retorna el valor de insertarNodo(x, nodo.izq), y si es mayor se retorna el valor de insertarNodo(x, nodo.der).

El método insertar(int x) simplemente le asigna al nodo raíz el valor insertarNodo(x, raiz).

El método buscarNodo(int x, NodoABB nodo) hace las mismas comparaciones. Si el nodo es nulo se retorna null, si x es igual al dato del nodo parámetro, se retorna dicho nodo, si x es menor al dato del nodo se retorna el valor buscarNodo(x, nodo.izq), y si es mayor se retorna buscarNodo(x, NodoABB nodo.der).

Luego el método buscar(int x) retorna el valor booleano de la sentencia buscarNodo(x, raíz) != null.

Clase AVL:

La clase AVL también posee una clase auxiliar NodoAVL, con un campo más que NodoABB llamado altura, y el campo principal de la clase AVL es el NodoAVL raiz.

El método de búsqueda es idéntico al del ABB normal, en cambio el método de inserción es diferente. Luego de hacer la inserción como un ABB normal, se verifica que la diferencia de las alturas de los subárboles izquierdo y derecho sean menores que 2. En caso contrario se violó la condición de equilibrio y se analiza que algoritmo de rotación corresponde. Si la inserción fue en el subárbol izquierdo del subárbol izquierdo se realiza una rotación simple, si fue en el subárbol derecho del subárbol izquierdo se realiza una rotación doble. En el caso de sobrecarga del subárbol derecho las rotaciones son análogas (casos espejo).

Luego de la inserción y reequilibrado se le asigna al árbol su altura correspondiente.

Clase ARN (Árbol Rojo-Negro):

Esta clase también posee una clase auxiliar llamada NodoARN, con los mismos campos de NodoABB y un campo para el color (boolean) y un campo para la altura.

El método de búsqueda de esta clase es idéntico al de las clases anteriores.

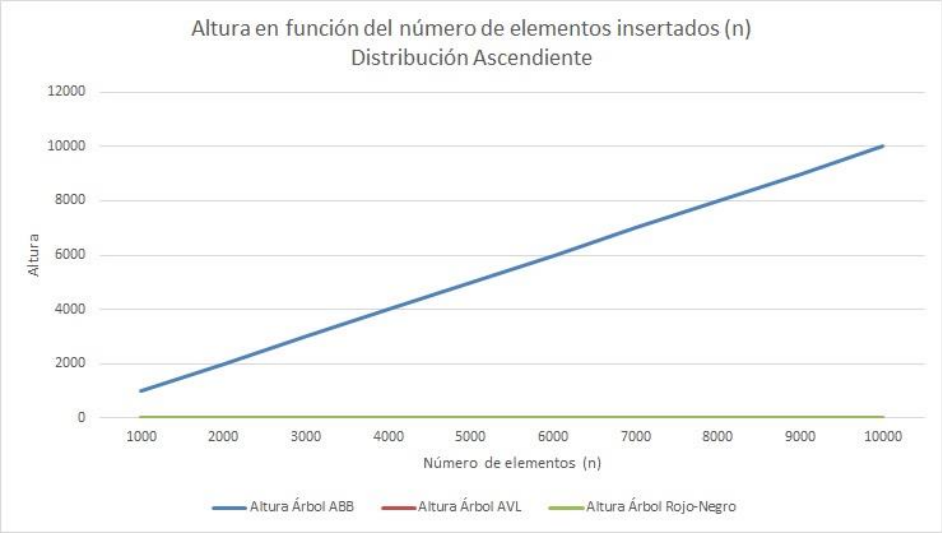
El método de inserción es diferente. Se inserta el nodo al igual que en la clase ABB, luego el procedimiento depende de los colores de los nodos izquierdo y derecho del árbol donde se hizo la inserción. Si el nodo derecho es rojo y el nodo izquierdo es negro, el árbol se rota hacia la derecha con una rotación simple. Si el nodo izquierdo y el nodo izquierdo del nodo izquierdo son rojos, el árbol se rota hacia la izquierda con una rotación simple. Si el nodo izquierdo y el derecho son rojos, se cambian los colores ejecutando la instrucción cambiarColores(nodo). Luego se le asigna al nodo su nueva altura y termina el método de inserción.

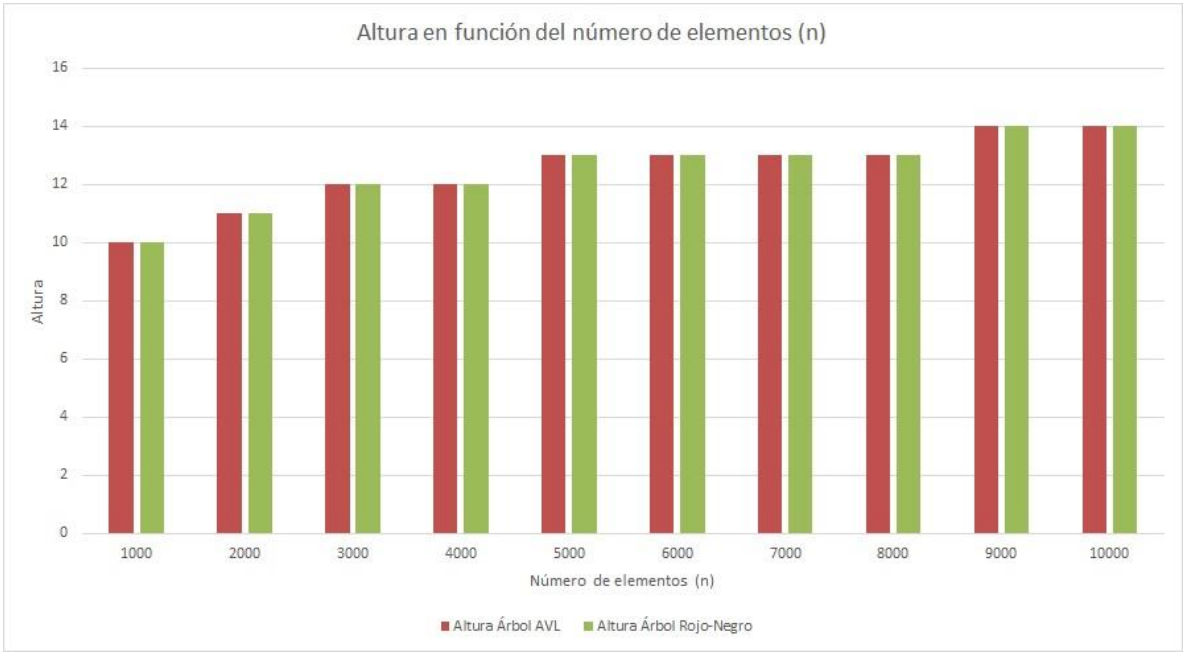
El método cambiarColores(nodo) cambia el color del nodo, de su hijo izquierdo y de su hijo derecho. Las rotaciones de este árbol son las mismas que las del AVL.

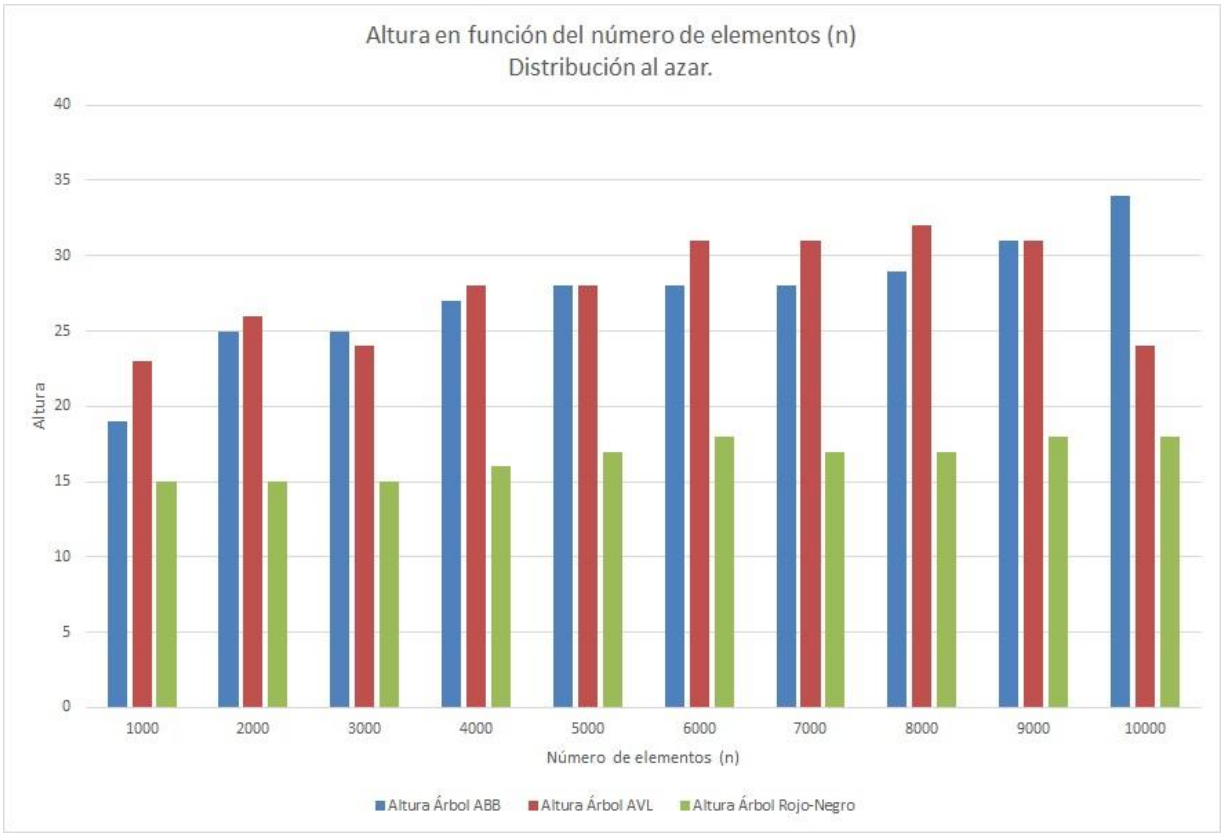
Resultados y Conclusiones:

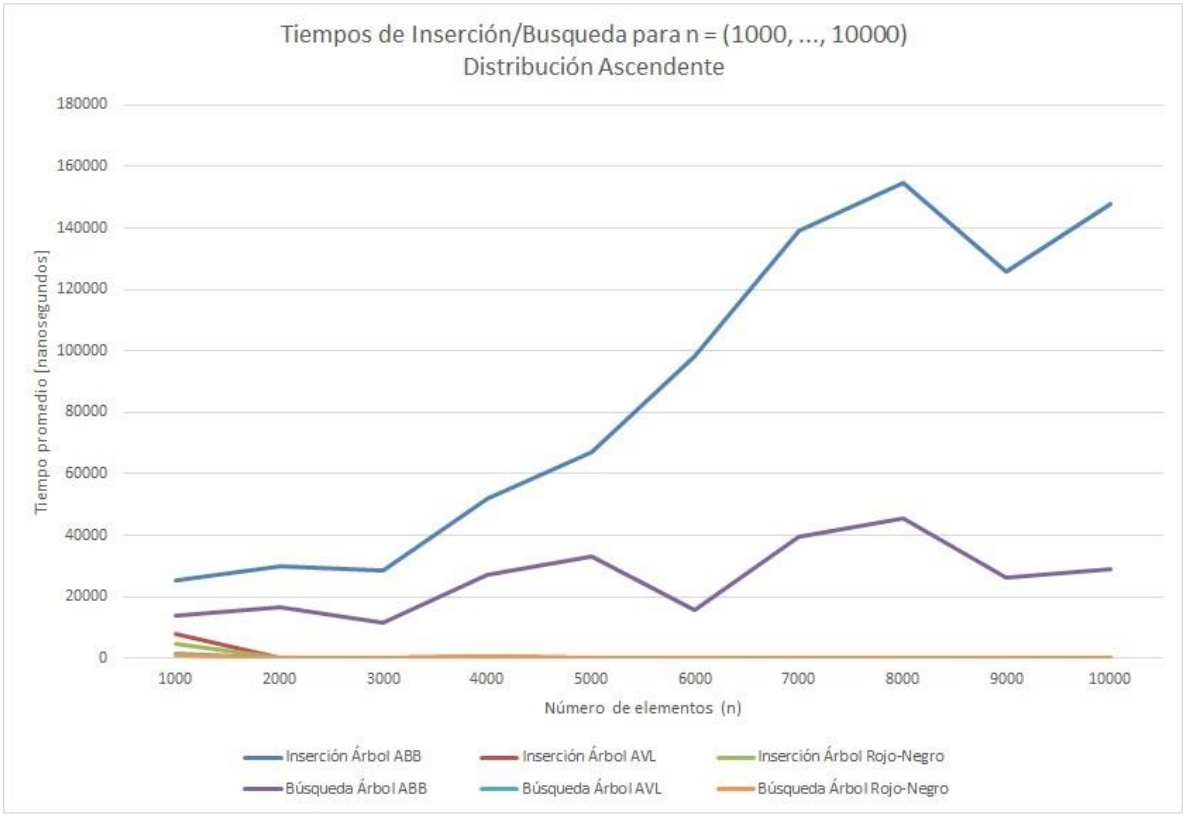
Se probaron las inserciones en los tres árboles variando la cantidad de elementos (n) a insertar entre {1000, 2000, …, 10000} y la distribución de dichos elementos, ascendente (empezando desde 1) y al azar. Las inserciones y búsquedas se hicieron iterando con un ciclo for, y se midieron los tiempos en nanosegundos gracias al método System.nanoTime(), que retorna el tiempo actual del sistema en nanosegundos, almacenado en un long.

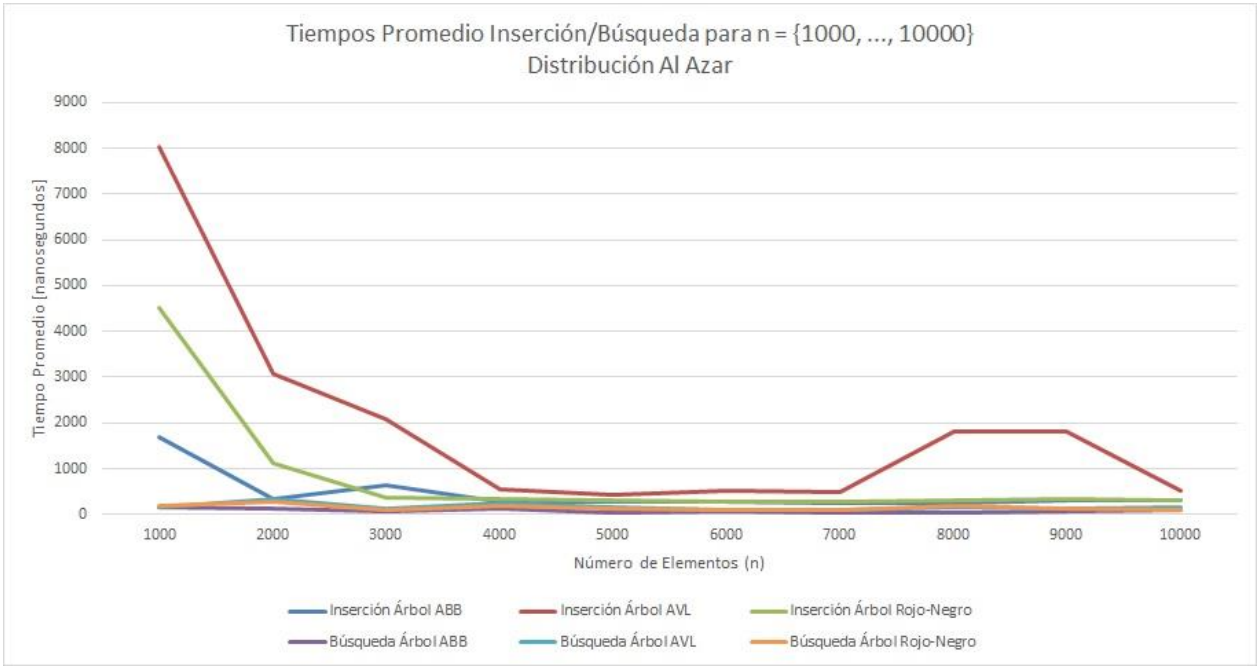
A continuación se adjuntan gráficos de la altura final de los árboles en función de n, y de los tiempos promedio de búsqueda e inserción en función de n:

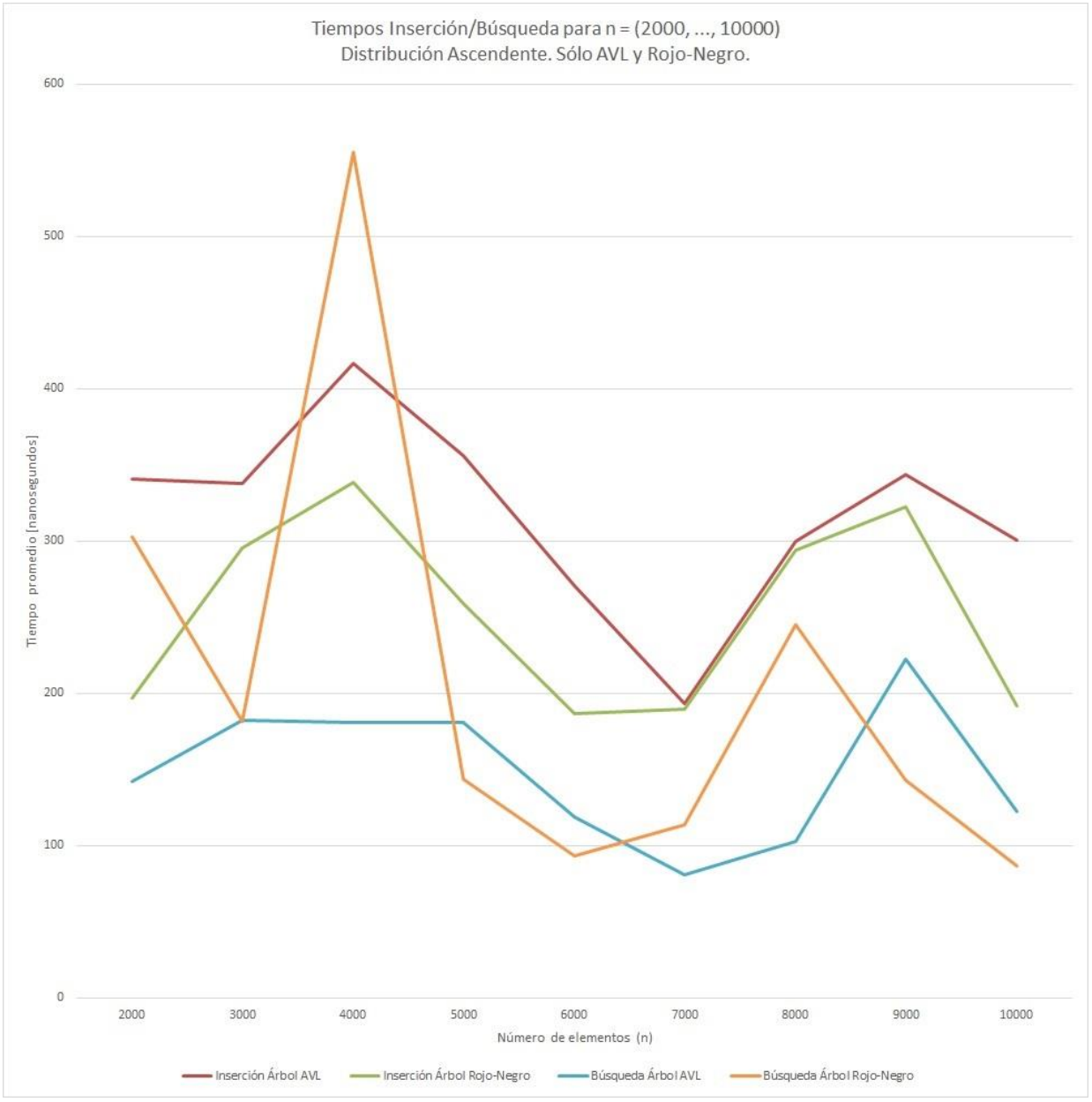


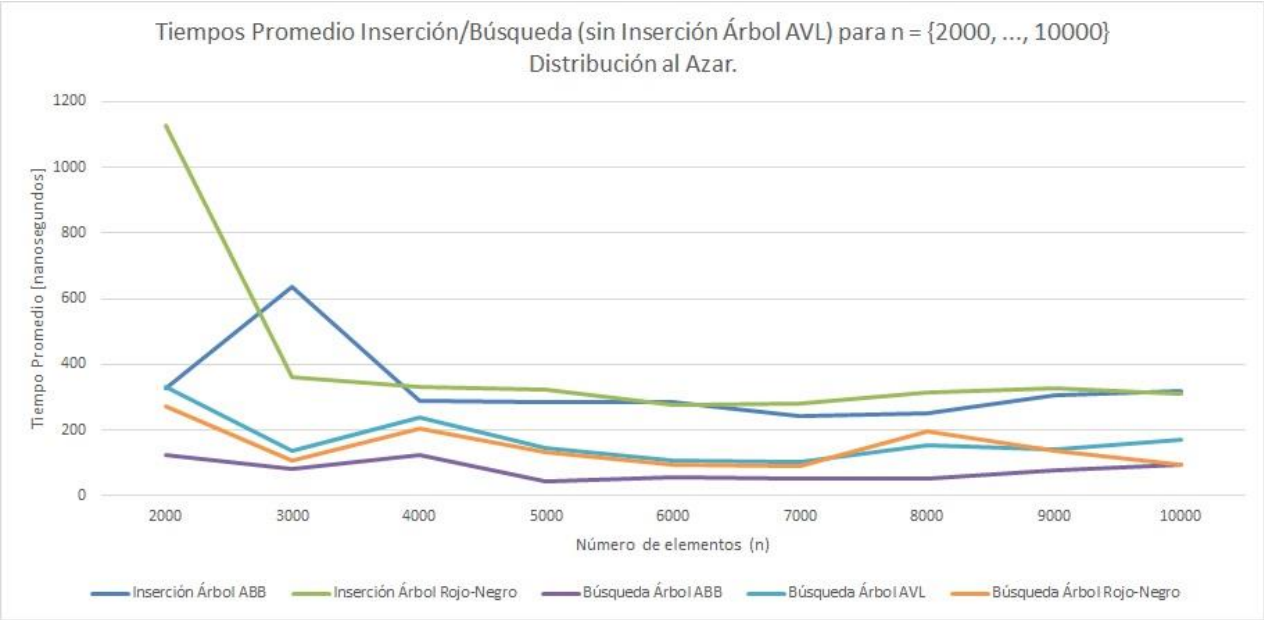

Altura en función del número de elementos (n)
16
14
12
10
8
6
4
2
0
Altura
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Número de elementos (n)
Altura Árbol AVL
Altura Árbol Rojo-Negro


Altura en función del número de elementos (n)
Distribución al azar.
40
35
30
25
20
15
10
5
0
Altura
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Número de elementos (n)
Altura Árbol ABB
Altura Árbol AVL
Altura Árbol Rojo-Negro


Tiempos de Inserción/Busqueda para n = (1000, ..., 10000)
Distribución Ascendente
Tiempo promedio [nanosegundos]
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Número de elementos (n)
Inserción Árbol ABB
Inserción Árbol AVL
Inserción Árbol Rojo-Negro
Búsqueda Árbol ABB
Búsqueda Árbol AVL
Búsqueda Árbol Rojo-Negro


Tiempos Promedio Inserción/Búsqueda para n = {1000, ..., 10000}
Distribución Al Azar
Tiempo Promedio [nanosegundos]
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Número de Elementos (n)
Inserción Árbol ABB
Inserción Árbol AVL
Inserción Árbol Rojo-Negro
Búsqueda Árbol ABB
Búsqueda Árbol AVL
Búsqueda Árbol Rojo-Negro


Tiempos Inserción/Búsqueda para n = (2000, ..., 10000)
Distribución Ascendente. Sólo AVL y Rojo-Negro.
Tiempo promedio [nanosegundos]
600
500
400
300
200
100
0
2000
3000
4000
5000
6000
7000
8000
9000
10000
Número de elementos (n)
Inserción Árbol AVL
Inserción Árbol Rojo-Negro
Búsqueda Árbol AVL
Búsqueda Árbol Rojo-Negro



- Altura: Para la distribución ascendiente la altura del ABB es igual a n, mientras que las alturas del AVL y del Árbol Rojo-Negro son iguales para todo n, y mucho menores que la del ABB. Para la distribución al azar en promedio la mayor altura es la del AVL, luego la del ABB y finalmente la del Árbol Rojo-Negro. Para la distribución al azar en promedio la altura del AVL es la mayor, seguida por la del ABB, siendo la del Árbol Rojo-Negro la menor.
- Tiempo promedio de inserción: Para la distribución ascendiente el tiempo promedio del ABB es mayor que el de los otros árboles, lo que hace al ABB el árbol menos eficiente para la inserción con esta distribución, y el tiempo promedio del AVL es mayor que el del Árbol Rojo-Negro, esto hace al Árbol Rojo-Negro el más eficiente de los tres para la inserción, con esta distribución. Para la distribución al azar el tiempo promedio del AVL es mayor, lo que lo hace el más ineficiente para la inserción con esta distribución. El tiempo del ABB y del Árbol Rojo-Negro son similares para n mayor o igual a 4000.
- Tiempo promedio de búsqueda: Para la distribución ascendiente el tiempo promedio del ABB es el mayor, lo que lo hace el árbol menos eficiente para las búsquedas con esta distribución. En promedio el tiempo del AVL es menor al del Árbol Rojo-Negro, lo que hace al AVL el árbol más eficiente para las búsquedas con este tipo de distribución. Para la distribución al azar el tiempo promedio del AVL y Árbol Rojo-Negro son similares, y el del ABB un poco menor.
- Luego para la distribución ascendiente el ABB es el árbol menos eficiente, el Árbol Rojo-Negro es el más eficiente para las inserciones y el AVL el más eficiente para las búsquedas. Esto es de esperar, ya que el AVL realiza en promedio más rotaciones post-inserción que el Árbol Rojo-Negro, por lo que se demora más en insertar, pero dado que su auto-balanceo es más riguroso las búsquedas son más eficientes.
- Para la distribución al azar el ABB y el Árbol Rojo-Negro poseen una eficiencia similar de inserción y mayor a la del AVL, y para la búsqueda el ABB es el más eficiente. Esto no es de esperar y puede deberse a un posible error de implementación en el AVL o el Árbol Rojo-Negro.

- Las alturas del AVL y del Árbol Rojo-Negro no están acotadas por log n, lo cual también puede deberse a un posible error de implementación. La cota del tiempo O(log n) si se cumple.