Escuela de Ingeniería
Departamento de Computación
Análisis y Diseño de Algoritmos (TC2017)
Profesor: Dr. Vicente Cubells Nonell

Tecnológico de Monterrey

Problema Práctico No. 1

Título	**Árboles balanceados**
Aprendizaje esperado (objetivo)	El alumno aprenderá a programar diferentes árboles de búsqueda utilizando Programación Genérica y aplicará los conceptos de medición de tiempo de ejecución y complejidad computacional para comparar y analizar los resultados obtenidos al buscar diferentes números sobre las estructuras implementadas.
Instrucciones	Implemente en C++ una clase genérica por cada uno de los tipos de árboles que se indican a continuación: AVL, B, Rojinegro y 2-3. Cada clase debe incluir las siguientes operaciones: • Insertar • Borrar • Buscar • Imprimir ordenado ascendentemente • Imprimir ordenado descendentemente • Obtener la altura de cualquier nodo • Obtener la profundidad de cualquier nodo • Obtener el nivel de cualquier nodo Posteriormente, diseñe y programe una aplicación que haciendo uso de las clases anteriores, muestre el funcionamiento de cada tipo de árbol. A continuación, realice un programa donde defina una población de 100 000 números aleatorios y con la misma población genere un árbol de cada tipo de los cuatro programados con anterioridad. Realice la búsqueda de 10 elementos aleatorios (que se encuentren en la población de números generados) en cada árbol y compare la eficiencia de los algoritmos de búsqueda utilizados por cada árbol, a partir de las mediciones del tiempo que demora cada búsqueda. Todas las mediciones deben realizarse en una Raspberry Pi.. Genere una gráfica (en Google Sheets) comparativa con los tiempos de ejecución de la búsqueda de cada uno de los 10 números en cada uno de los cuatro árboles implementados. Debe ser una gráfica de barras horizontales multi-series (una serie por cada número buscado). Analice e interprete los resultados alcanzados. Confeccione un documento en Google Docs donde incluya la tabla de resultados, las gráficas comparativas y su interpretación personal de los resultados obtenidos. Suba a la plataforma el archivo con los resultados e incluya en el mismo la liga al repositorio de GitHub dentro de la clase (https://classroom.github.com/g/ZOE2NcG8) que contenga todos los códigos de los algoritmos programados.

	No se aceptan trabajos fuera de fecha ni por correo electrónico. En ambos casos la calificación de la tarea será 0 puntos. En esta actividad cada miembro del equipo evaluará (de 1 a 10) la participación de los demás miembros en base al tiempo y trabajo realizado por cada uno en la solución de la actividad. Esta coevaluación deberá ser objetiva y con un espíritu constructivo. Esta coevaluación aportará el 10% de la calificación de la actividad.
Forma de trabajo	En equipos de 2 estudiantes
Recursos	Foros de información en Internet Wikipedia (http://www.wikipedia.org) Códigos de árboles vistos en la materia Estructura de Datos Computadora
Tiempo estimado	10 horas
Criterios de evaluación	La implementación de cada árbol (4) utilizando Programación Genérica y con las operaciones indicadas tendrá un valor de 16 puntos (2 puntos por operación). La programación de un ejemplo que demuestre el uso de cada clase (4) así como de las diferentes operaciones tendrá un valor de 4 puntos. Medir el tiempo de ejecución de cada búsqueda (10) en cada árbol (4) tendrá un valor de 0.25 puntos. Generar la gráfica que incluya los tiempos de ejecución de todos las búsquedas tendrá un valor de 2 puntos. Analizar e interpretar los resultados correctamente tendrá un valor de 3 puntos. La coevaluación tendrá un valor de 5 puntos.
Valor de la actividad	20% de la calificación del primer parcial

Respuestas

Repositorio de GitHub:

https://github.com/tec-csf/TC2017-P1-Otono-2019-equipo-algoritmos

Tabla con los resultados de las mediciones:

Tabla de resultados a completar (*tiempo en ms*)							
No.	**Tipo de árbol**		**[1]**	**[2]**	**[3]**	**[4]**	**[5]**
1	**AVL**	*RPi*	1.63241	1.5432	1.239821	1.45612	1.2739721
2	**B**	*RPi*	0.1960410	0.181570	0.181580	0.1829	0.187220
3	**Rojinegro**	*RPi*	0	5	6	0	6
4	**2-3**	*RPi*	1	1	5	5	1

Tipo de árbol		**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**AVG**
AVL	*RPi*	1.087239	1.923742	1.27382	1.28721	1.00972	**1,37273**
B	*RPi*	0.1868479	0.1844460	0.1851730	0.1808480	0.1850010	**0,1851627**
Rojinegro	*RPi*	1	5	7	5	5	**4**
2-3	*RPi*	6	8	5	5	6	**4.3**

Interpretación de los resultados:

- **Árboles AVL:** Son árboles que se balancean por sí solos, pues se aseguran que para cualquier nodo del árbol, la diferencia entre la altura del subárbol derecho menos la altura del subárbol izquierdo, no exceda a la unidad. Con esto se asegura que la altura de un árbol AVL sea O(log N). En general, la altura de estos árboles es de aproximadamente 1.44 * log N.
- **Árboles B:** La mayoría de las operaciones de estos árboles requieren de O(h) accesos al archivo, donde h equivale a la altura del árbol. Como en cada uno de los nodos se tienen N + 1 elementos, se debe de realizar una búsqueda binaria de O(log N + 1). Al tener más hijos en sus nodos, se incrementa la búsqueda por nodo, pero se minimiza la cantidad de nodos por visitar. Su complejidad resulta en O(logN)
- **Árboles 2-3:** Se caracterizan porque todos los caminos de la raíz hasta la hoja son de exactamente la misma altura, pero permiten que los nodos internos tengan de 2 a 3 hijos. En promedio, tienen una complejidad de O(log N).

- **Árboles Rojinegros:** Pueden ser vistos como una variación de los árboles 2-3. La manera de implementar la búsqueda en los árboles Rojinegros, es muy similar a la de los Árboles Binarios de Búsqueda (ABB), aunque es más rápida que los ABB por el simple hecho de estar más balanceados. La altura de los Rojinegros pueden llegar a ser hasta de 2 * log N.

Nuestra tabla de resultados, nos muestra que, en promedio, el tiempo de ejecución de búsqueda en los árboles B es mucho más rápida que la de los otros 3 algoritmos (0.1851627ms vs 1.37237ms vs 4ms vs 4.3ms). Esto es muy probablemente por ser el único que lee los datos de un archivo, así como también posee la característica de tener pocos niveles y ser muy anchos, ya que al minimizar la altura del árbol, también se minimizan los accesos al disco (o al archivo). Comparando los árboles B con os AVL, se puede concluir que los AVL no están diseñados para guardar grandes cantidades de datos ya que usan memoria dinámica y apuntadores hacia el siguiente bloque de memoria.

Los tiempos de ejecución entre los árboles Rojinegros y 2-3 fueron muy similares, lo cual puede ser que de deba a su implementación tan similar. En general la altura de los AVL es un poco menor que la de los Rojinegros, y los AVL se encuentran mejor balanceados, la búsqueda en los AVL suele ser un poco más rápida.

Aún así, todos los árboles son realmente eficientes, pues al tener una complejidad de O(log N), la cantidad de comparaciones es muy baja. Por ejemplo:

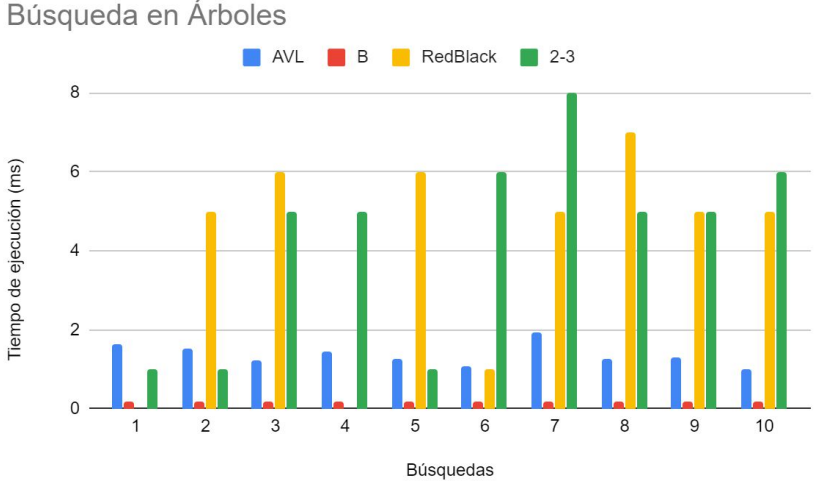
$$\text{si } n = 1\ 000\ 000 \rightarrow \log_2 1\ 000\ 000 = 19.9315685693$$

En este caso se tendrían que realizar ~19 comparaciones para encontrar, insertar, eliminar nodos de un árbol.

Simplemente se debe de tomar en cuenta para qué se estarían utilizando los árboles para poder aprovecharlos al máximo e intentar implementar los algoritmos de la manera más óptima posible.

Es importante recalcar que el tiempo de ejecución de estos algoritmos dentro de una computadora tan pequeña como lo es la Raspberry Pi, varían mucho dependiendo de la implementación que se les de. Por lo que en nuestros códigos, quizás se podría haber llegado a una implementación mucho más óptima que la que se realizó.

Gráfica comparativa



Coevaluación:

Me parece que mi compañero Ricardo trabajó muy bien, pues estuvo al pendiente de organizarnos para entregar el proyecto en tiempo y en forma. Le asigno un 10.