Universidad del Valle de Guatemala *Oscar Castañeda- Carnet 11086*
Algoritmos y Estructura de Datos – Sección 20 *Luis Valdeavellano – Carnet 11218*

Hoja de Trabajo 8:
Implementación de BST y Hash

Para la realización de esta hoja de trabajo, se obtuvieron del sitio http://dept.cs.williams.edu/JavaStructures/Software.html las implementaciones realizadas por Duane A. Bailey, autor del libro de texto del curso "Java Structures", para las estructuras de datos a utilizar, las cuales eran el Red Black Tree, el Splay Tree y la Hash Table. Además, se utilizó la clase SimpleSet ya proveída con el programa a mejorar para esta hoja, la cual trabajaba con un ArrayList, y se añadió una clase del Java Collection Framework (JCF). Dicha clase era LinkedHashMap, una implementación muy similar a una Hash Table, con la diferencia de que sus entradas se encuentran unidas por una lista doblemente encadenada.

Una vez mejorado el programa, se midió el tiempo utilizado y el uso de memoria con el profiler de Netbeans. Las capturas de pantalla de los resultados se encuentran en la carpeta "Resultados Profiler" del archivo comprimido adjuntado con esta tarea (también se encuentra en la misma carpeta en el repositorio GIT en Google Code, cuya dirección es http://code.google.com/p/ht7-bst-splay-hash/). A continuación se resumirán dichos resultados en tablas:

Implementación	Tiempo para añadir todas las palabras (Método add) [ms]	Tiempo para buscar las palabras halladas en la estructura (Método get) [ms]	Tiempo Total [ms]
Linked Hash Map (de JCF)	65.3	5.77	71.07
Red Black Tree	81.2	1.7	82.9
Splay Tree	147	28.5	175.5
Hash Table	221	26.9	247.9
SimpleSet (ArrayList)	15.1	7923	7938.1

Tabla 1 *Resultados del profiler para el tiempo utilizado por cada una de las implementaciones, ordenadas del menor al mayor tiempo total*

Implementación	Bytes utilizados
Red Black Tree	2230864
SimpleSet (ArrayList)	2566192
Splay Tree	2764856
Linked Hash Map (de JCF)	2813376
Hash Table	2915712

Tabla 2 *Resultados del profiler para la memoria utilizada para cada una de las implementaciones, ordenadas del menor al mayor uso*

Puede observarse de la tabla 1 que la implementación de WordSet que toma un menor tiempo total es la basada en Linked Hash Map, de JCF, a pesar de que no cuenta con el menor tiempo individual ni para cargar las palabras ni para buscarlas. Linked Hash Map es, prácticamente, una Hash Table optimizada al contar con una lista doblemente encadenada, por lo que se conoce que su complejidad en tiempo, tanto para insertar un elemento como para buscar, es O(1) (es decir, constante). Nótese también que, por ser una versión mejorada de Hash Table, esta clase de JCF necesita de un menor tiempo que la Hash Table implementada por Bailey.

Sin embargo, hay otros casos que resultan interesantes de analizar. Uno de ellos es el ArrayList utilizado por SimpleSet, el cual presenta el menor tiempo para agregar todas las palabras a la estructura. Esto sucede porque no solo tiene una complejidad amortiguada de O(1) para añadir un elemento al final de sí mismo, sino que además no necesita realizar ninguna acción adicional para agregar un elemento, no como sucede en los Splay y Red Black Tree, los cuales deben de ajustarse luego de añadir un elemento para quedar en un estado consistente, o como las Hash Table, que deben de aplicar una función de hash para determinar la futura ubicación del elemento a añadir. No obstante, la complejidad del Array List para buscar un elemento es O(n) y el tiempo utilizado para dicha acción resulta demasiado elevado.

El otro caso interesante se observa en el Red Black Tree, el cual presenta un tiempo total muy bueno (tan solo se diferencia por 11.83 ms del tiempo logrado por el Linked Hash Map). Lo sorprendente en este caso es el tiempo de búsqueda, el cual es de apenas 1.7 ms, siendo alrededor del 30% del tiempo utilizado por Linked Hash Map. Si bien la complejidad del Red Black Tree es O(log n) para la búsqueda (que no es tan buena como la complejidad de O(1) del Linked Hash Map), aun así el tiempo registrado por el profiler para tal acción en dicha clase es considerablemente menor frente a los tiempos presentados por las demás implementaciones, convirtiendo a este árbol en una muy buena opción. No obstante, como ya se ha mencionado antes, añadir un elemento al Red Black Tree es más complicado de hacer que en otras implementaciones como la Hash Table, puesto que se debe de mantener a la estructura en un estado consistente, pero a pesar de esto, la complejidad de la inserción sigue siendo de O(log n) en tiempo, la cual es una buena complejidad.

Por otra parte, al analizar los resultados para la memoria utilizada por cada una de las implementaciones, se puede observar que ambas implementaciones de Hash Table (la de Bailey y la de JCF con Linked Hash Map) son las que utilizan una mayor cantidad de memoria. Esto es de esperar, puesto que dichas estructuras reservan un espacio en memoria del cual no pueden

utilizar un porcentaje mayor que cierta capacidad de carga, para poder ofrecer una mejor complejidad en el tiempo de sus operaciones. Además, la implementación que utiliza una menor cantidad de Bytes es el Red Black Tree, teniendo una diferencia de 684,848 Bytes con el Hash Table y de 582,512 con el Linked Hash Map, su competidor en el desempeño en cuestiones de tiempo. Dicha diferencia se incrementará al ir añadiendo más palabras a la estructura, puesto que el Hash Table y Linked Hash Map se expandirán para no sobrepasar su carga máxima.

Por todo lo anterior, se concluye que la mejor implementación es el Red Black Tree, puesto que ofrece un menor uso de memoria y, a pesar de que su complejidad para añadir y buscar un elemento es de O(log n) (la cual no es tan buena como la complejidad de O(1) de las Hash Tables), el tiempo en que realiza la búsqueda es significativamente menor al de las demás implementaciones. Incluso, se puede observar que, para fines de este programa, si el texto es más largo que el utilizado en estas corridas, el tiempo de añadir todas las palabras al árbol seguiría siendo el mismo, pero dada la complejidad y tiempo registrado para la búsqueda, se completaría todo el análisis del texto en un tiempo total incluso menor que el que haría Linked Hash Map (puesto que cada una de las búsquedas en el Red Black Tree toma un tiempo menor que en el Linked Hash Map).

Como un comentario final, debe de observarse el caso del Splay Tree. El Splay Tree no llegó a destacar tanto como el Red Black Tree en ninguna de las mediciones (de hecho, siempre ocupó el puesto medio). Esto sucede porque dicho árbol está pensado para dar una complejidad O(log n) en tiempo, siempre y cuando solamente se accedan a aproximadamente 10% de los datos almacenados en el, de manera repetitiva. Esto no se cumple de la mejor forma al buscar palabras, ya que si es cierto que existen palabras que son utilizadas constantemente, existen muchas otras que no lo son, y estas vienen a impactar negativamente sobre la complejidad amortizada de esta estructura.