Universidad Central “Marta Abreu” de Las Villas
Facultad de Matemática, Física y Computación

Trabajo para optar por Título de
Licenciado en Ciencia de la Computación

Implementación de variantes complejas de árboles binarios en apoyo a la docencia

Autor
Roilyn Miguel Quiñones Méndez

Tutor
Dr. Daniel Gálvez Lio

2011-2012

Declaración Jurada

El que suscribe, Roilyn Quiñones Méndez hago constar que el trabajo titulado: **Implementación de variantes complejas de árboles binarios en apoyo a la docencia** fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de los estudios de la especialidad de Licenciatura en Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del tutor Firma del jefe del Laboratorio

Fecha: 27/09/2012

Dedicatoria:

A mis padres, amarme y apoyarme en todas las etapas de mi vida.

A mi hermano, por ser mi mano derecha.

A Rosy, por existir en mi vida.

A Landy, te quiero mucho.

Agradecimientos:

A mi mamá, por ser única.
A mi papá, por sacarnos adelante en la vida.
A mi hermano, por la infancia tan linda que me dio.
A Rosy, por su amor y compañía incondicional, sin ti no hubiese llegado a la meta.
A mi abuela, por su paciencia.
A mi tía Marlen y a mi primo Víctor por su apoyo.
A Daniel Gálvez, por ser un excelente tutor.
A mis suegros y su familia, por aceptarme como uno más.
A toda mi familia de Santi Spíritus, por el apoyo a pesar de la lejanía.
A mis amigos del pre, por crecer junto a mí.
A mis amigos de la universidad, por su compañía en estos 5 maravillosos años.
A mis vecinos (todos), por el día a día.
A Odalis y Osmani, por adoptarme.
A toda mi familia, por existir.

Resumen

La especialidad de Licenciatura en Ciencia de la Computación contempla entre sus objetivos fundamentales la formación de un profesional capaz de enfrentarse a cualquier problemática referida al ámbito de la programación. Dos de las asignaturas de la disciplina de Programación son Estructuras de datos y Algoritmos I y II, en particular en la primera asignatura se estudian un grupo de estructuras de tipo árbol binario de búsqueda de gran importancia en Computación, pero por su complejidad en ocasiones no son impartidas con todos los elementos indicados en el plan de estudio.

Este trabajo se centra en el estudio de 4 tipos de árboles: los árboles binarios de búsqueda, los árboles AVL, los árboles rojo y negro y los árboles B. Para estos tipos de árboles se desarrollan implementaciones en Java que permiten ser empleados en la docencia como apoyo al proceso de aprendizaje de los estudiantes bajo la orientación del profesor. Las herramientas desarrolladas están formadas por dos paquetes y una aplicación de escritorio que permite visualizar las operaciones básicas para cada tipo de árbol.

Tabla de contenido

Introducción

La especialidad de Licenciatura en Ciencia de la Computación de la Facultad de Matemática, Física y Computación en la Universidad Central "Marta Abreu" de las Villas, contempla entre sus objetivos fundamentales la formación de un profesional capaz de enfrentarse a cualquier problemática referida al ámbito de la programación.

Los árboles constituyen uno de los puntos elementales del contenido a dominar. Los estudiantes deben saber dominar los diferentes tipos de árboles, las especificidades de cada cual, así como las operaciones fundamentales. Las estructuras arbóreas tienen muchas aplicaciones en el área de la programación específicamente en el ordenamiento y almacenamiento de las estructuras de datos abstractas.

Cada uno de los tipos de árboles tiene sus especificidades, características que los diferencian del resto y los lleva a tener determinada área de utilización. Los Árboles Binarios de Búsqueda (ABB), por ejemplo, resaltan por su método de búsqueda, es un método de búsqueda dinámico y eficiente considerado como uno de los fundamentales en Ciencia de la Computación. Aquí también sobresale el AVL, con su condición de equilibrio y el rojinegro (RN) donde cada nodo tiene un atributo de color cuyo valor es rojo o negro. Entre los principales ejemplos de árboles se encuentran también los Árboles B, los cuales son árboles tipo "multiway".

Partiendo de la necesidad del estudiante de especialidades de computación en la profundización de estos contenidos, proponemos el siguiente **Objetivo General**:

Implementar las variantes de árboles más complejas que se indican en el programa de las asignaturas de Estructuras de Datos y Algoritmos I y Estructuras de Datos y Algoritmos II para facilitar su estudio y asimilación por parte de los estudiantes de ambas carreras.

Para dar cumplimiento a este Objetivo general trazamos entonces los **Objetivos Específicos**:

- Analizar las orientaciones del programa de las asignaturas de Estructuras de Datos del plan de estudio de D de la carrera de Ciencia de la Computación en lo relativo al estudio de variantes de árboles complejas.
- Proponer soluciones computacionales para implementar las variantes de árboles complejas indicadas en los planes de estudios.
- Implementar las soluciones computacionales propuestas.

- Proponer el uso de las herramientas desarrolladas dentro del programa de las asignaturas de Estructuras de Datos de la carrera de Ciencia de Computación.

Esta investigación se plantea para dar respuesta a determinadas necesidades teóricas y prácticas tanto de profesores de las asignaturas Estructuras de Datos y Algoritmos I y II, como de sus respectivos estudiantes. Es decir, que el software obtenido como resultado final, debe constituir un medio de enseñanza en la clase, así como de apoyo al estudio individual del estudiante de Informática y de Ciencia de la Información.

Hasta el momento los códigos de las principales operaciones con árboles no se encontraban totalmente en la bibliografía fundamental de la asignatura, ni tampoco en las Conferencias y Clases Prácticas planificadas por el profesor. En el software el estudiante no solo puede acceder a estos códigos, sino además que las operaciones se muestran de manera dinámica y atractiva para que el estudiante pueda interactuar con el contenido.

Capítulo I: ESTUDIO DE ÁRBOLES AVANZADOS EN LAS CARERAS DE CIENCIAS DE LA COMPUTACIÓN E INGENIERÍA INFORMÁTICA

Las comunicaciones rigen hoy la dinámica de una humanidad cada vez más dependiente de las tecnologías, en especial la computación, presente en los ámbitos educacionales, en los aeropuertos, en la medicina y, por supuesto, en las esferas de esparcimiento. Por ello, estas acciones "exigen de la actividad de desarrollo de software un carácter cada vez más flexible, versátil y creativo." (MES, 2007:81)

Esa es precisamente la misión de las universidades donde se imparten especialidades relacionadas con la computación, ya que entre las actividades más importantes de este tipo de especialista se encuentra la asimilación, evaluación, aplicación y desarrollo de aplicaciones de software.

En el desarrollo del Capítulo I se analizará la asignatura Estructura de Datos en las especialidades Licenciatura en Ciencias de la Computación e Ingeniería Informática. Posteriormente se realizará una síntesis teórica sobre el tema árboles, haciendo un especial énfasis en los Árboles Binarios de Búsqueda y sus diferentes tipos, así como los árboles B y otros tipos de árboles. De cada uno de los árboles se especificará su definición, sus principales características y sus principales operaciones.

1.1 Plan de Estudios D

La Facultad Matemática, Física, Computación de la Universidad Central "Marta Abreu" de las Villas ofrece la especialidad Licenciatura en Ciencias de la Computación. Desde su primer año de estudios la disciplina *Programación e Ingeniería de Software* tiene un carácter preponderante, razón por la cual resulta la de más peso en cuanto a horas. Además, es la que presenta más asignaturas comunes a "incluir en los planes específicos de cada institución y con más presencia desde el inicio del programa de estudios" (MES, 2007:80).

En el Plan de Estudios D de la disciplina *Programación e Ingeniería de Software* para la carrera Ciencias de la Computación se definen sus principales objetivos: "La disciplina y todas las asignaturas deben hacer un balance adecuado entre lo conceptual y práctico, entre lo abstracto y lo concreto, entre los aspectos más formales y los más intuitivos balanceando e interiorizando siempre en los alumnos el doble papel científico-tecnológico que tiene la especialidad". (MES, 2007:88)

Dentro de la disciplina se pueden reconocer las asignaturas Estructura de Datos y Algoritmos I y II (EDA I y II). En la carrera Ciencia de la Computación comienza a impartirse en el primer semestre de 2do año, con un total de 64 horas, donde se crea en el profesional la habilidad para ofrecer una solución computacional a problemas de cualquier rama.

El mencionado Plan D establece entre el sistema de conocimientos de la asignatura el tema Árboles, entre ellos: "Árboles. Árbol ordenado. El TDA Árbol Binario. Algoritmo de Compresión de Huffman. Cola con prioridad. Árbol parcialmente ordenado (HEAP binario). Árbol Binario de Búsqueda (ABB). El AVL como ejemplo representativo de ABB balanceado por alturas. El Árbol Rojo–Negro (R-N), como ejemplo representativo de árbol balanceado por colores. El 2-3 Árbol. Definición y propiedades del mismo. Análisis de la altura del árbol R-N y su incidencia en la complejidad algorítmica de las operaciones fundamentales: búsqueda, inserción y extracción. El Tree, como una forma de representación de conjuntos de cadenas de caracteres. El QuadTree, como una forma de representación de conjuntos de puntos en el plano. Introducción al Octree como una representación de conjuntos de puntos en el espacio. (MES, 2007:89)

1.1.1Ejecución de la asignatura Estructura de Datos I en el curso 2009-2010

La Facultad Matemática, Física, Computación de la Universidad Central "Marta Abreu" de las Villas ofrece la especialidad Licenciatura en Ciencias de la Computación. A partir del curso 2008-2009 comienza a aplicarse el Plan D. No es hasta el año siguiente que comienza a impartirse EDA con el nuevo Plan de Estudios.

En el Programa Analítico de la asignatura EDA I, se declara que el Tema IV referente a Árboles será impartido en 8 horas de Conferencias y 32 horas de Clases prácticas, de las cuales 8 son de laboratorio. Entre sus objetivos fundamentales está la necesidad de que el estudiante conozca las operaciones y recorridos de los árboles y sea capaz de implementar aplicaciones donde se utilicen estas estructuras

Entre los contenidos a impartir declara: Árboles. Árbol ordenado. El TDA Árbol Binario. Algoritmo de Compresión de Huffman. Cola con prioridad. Árbol parcialmente ordenado (HEAP binario). Árbol Binario de Búsqueda (ABB). El AVL como ejemplo representativo de ABB balanceado por alturas. El Árbol Rojo–Negro (R-N), como ejemplo representativo de árbol balanceado por colores.

Los temas fueron impartidos en cuatro conferencias. La Conferencia 5: “Árboles”, donde se tratan los árboles: definición y propiedades; árboles binarios: recorridos, operaciones básicas y complementarias, pero de las operaciones más importantes solo se explica (con el apoyo de un ejemplo) buscar y en cuanto a la implementación se explica el arreglo de padres y la lista enlazada (específicamente la lista multienlazada) La explicación de los árboles binarios contiene: definición, ejemplos, recorridos, así como la ejemplificación de su uso en el Código de Huffman. Las clases prácticas que complementan esta conferencia sí ejercitan todas las operaciones fundamentales de los árboles: buscar, eliminar e insertar.

La Conferencia siguiente, “Conjuntos dinámicos. Árboles binarios de búsqueda” abordan los siguientes tópicos: definición y la explicación de sus operaciones fundamentales, aunque solo la operación eliminar de los ABB y su implementación se imparte a través de la presentación de sus respectivos códigos.

Por otra parte, la conferencia 7: “Conjuntos Dinámicos con árboles balanceados. Árboles de búsqueda balanceados AVL”, explica su definición así como el Factor de equilibrio como propiedad fundamental de este tipo de árboles, y la importancia de las diferentes rotaciones para el sostenimiento de su principal condición. Por último en la conferencia 8: “Implementación de conjuntos con árboles Rojinegros”, se definen y se ofrece el código de implementación, pero solo se especifica en la operación insertar, sobretodo en la inserción descendente, tanto en la conferencia como en la CP.

El estudio y profundización de las estructuras arbóreas culmina en el segundo semestre de segundo año con EDA II a través del te4ma III: “Árboles B y variaciones” y el “TEMA IV: QuadTree y OctTree”, con base en el siguiente sistema de conocimientos:

- El Tree, como una forma de representación de conjuntos de cadenas de caracteres. El QuadTree, como una forma de representación de conjuntos de puntos en el plano. Introducción al Octree como una representación de conjuntos de puntos en el espacio.
- Métodos de Ordenación Externa.
- Almacenamiento en Memoria Externa. El Árbol B (B-Tree).

Para su desarrollo se imparte la Conferencia 6: “Árboles B”, donde se parte de la definición y caracterización de los Árboles 2-3 así como la explicación y los ejemplos de sus operaciones, para luego explicar qué son los árboles B y presentar el algoritmo de

buscar, eliminar e insertar, así como la especificar en las características de los árboles B+ y B*.

1.1.2Ejecución de la asignatura Estructura de Datos I en el curso 2010-2011

La asignatura EDA I fue impartida en el curso 2010-2011 con algunas transformaciones. Las conferencias referidas a los temas Árboles, ABB y AVL son iguales a las del curso procedente, sin embargo no se imparte el tópico Árboles Rojinegro. En este curso también se incluye el tema árboles en el segundo semestre, con la conferencia perteneciente al Tema IV: "Árboles B-Tres", donde se obvian algunos contenidos impartidos el curso anterior (Árboles 2-3) y se añaden otros, como el cálculo de la altura. Las Clases Prácticas "Eliminación en un B-Tree" e "Inserción en un B-Tree", complementan la conferencia. El programa analítico no presenta variaciones respecto al del 2009-2010.

1.1.3Ejecución de la asignatura Estructura de Datos I en el curso 2011-2012

El tercer curso de experiencia del Plan D para las asignaturas EDA I y II, tiene características diferentes a los años anteriores. Por ejemplo en EDA I se imparte ABB, AVL y Rojinegro[1], pero no se da la conferencia referida a los árboles de manera general. Ni en este semestre ni en EDA II, se tocan otros temas ausentes como Árboles B y los Quadtree y Octree.

O sea, que este último año de interés para la presente investigación no se logra igual profundización de contenidos del tema árboles, en comparación con los cursos anteriores.

1.2 Estructura de Datos en la carrera de Ingeniería Informática

En la especialidad de Ingeniería Informática, la asignatura Estructura de Datos también aplica el Plan de Estudios D. Los árboles o las estructuras arbóreas se encuentran entre sus temas fundamentales.

A partir del curso 2010-2011 se implementan algunos cambios orientados por la comisión nacional de la carrera. Estas transformaciones se basaron en los resultados negativos obtenidos por los estudiantes en exámenes diagnósticos, preguntas escritas,

[1] Las Conferencias son las mismas que las del Curso 2009-2010

orales, así como otras evaluaciones, por lo que se demostró la necesidad de que los alumnos contaran con una mayor cantidad de horas para una mejor asimilación, el contenido se mantendría igual.

Por tanto se decide tener dos asignaturas: Estructura de datos I y II. El total de horas supera en un 40% a las anteriores.

Con esta orientación hace dos cursos se trabaja en las asignaturas de Estructuras de Datos, en las que de manera similar a la carrera de Ciencia de la Computación, algunas operaciones básicas de los tipos de árboles mas complejos son estudiadas someramente.

1.3 Árboles Binarios de Búsqueda

Los árboles son una estructura de datos ampliamente utilizada muy a menudo en la computación. La mayoría de las definiciones coinciden en que está formada por "un conjunto de nodos y un conjunto de aristas que conectan pares de nodos" (Allen, 2004: 147). Otros textos simplemente la identifican como conjunto de nodos conectados (Gutiérrez, 2010). En "Estructuras de datos dinámicas" ofrecen el siguiente concepto:

"Árbol: estructura no lineal y dinámica de datos. Dinámica: puede cambiar durante la ejecución de un programa. No lineal: a cada elemento del árbol pueden seguirle varios elementos. Están formados por un conjunto de nodos y un conjunto de aristas que conectan pares de nodos." (ICT, 2012)

Otros textos ofrecen una conceptualización de árboles divididas en dos grandes ramas: la definición recursiva y la no recursiva. La primera versión los identifica como: "un árbol es o bien vacío o consiste en una raíz y cero o más subárboles no vacíos T_1, T_2, …, T_k, cada una de cuyas raíces está conectada por medio de una arista con la raíz. Una consecuencia de esta definición es que cada nodo de un árbol define un subárbol, que es aquel que le tiene por raíz. Se tiene así una identificación entre nodo y subárbol (…). En algunas clases de árboles, como los árboles binarios (…) se puede permitir que algunos de los subárboles sean vacíos." (Allen, 2004: 346-347)

Mientras que la definición no recursiva establece que un árbol es un conjunto de nodos y conjunto de aristas que conectan pares de nodos con las siguientes características:

- Se distingue un nodo raíz (no tiene padre).
- A cada nodo c (excepto la raíz) le llega una arista desde exactamente un nodo p diferente a c, al cual se le llama padre de c.

- Hay un único camino desde la raíz hasta cada nodo. La misma longitud del camino es su número de aristas. (ICT, 2012)

Los árboles permiten modelar diversas entidades del mundo real tales como, por ejemplo, el índice de un libro, la clasificación del reino animal, el árbol genealógico de un apellido, etc. (OWC, 2012: 143)

El texto *Estructuras de Datos en Java™ Compatible con Java™ 2*, Tomo I, describe los principales usos de los árboles en computación:

> *Casi todos los sistemas operativos almacenan los ficheros en estructuras que son árboles o similares. Por ejemplo, bajo DOS, VMS o Unix, los directorios se almacenan en los nodos internos que no son hojas del árbol, mientras que el resto de ficheros se almacenan en las hojas. (...) Una segunda aplicación de los árboles corresponde a los denominados árboles de expresiones (...). En el árbol sintáctico de una expresión, el valor de un nodo es el resultado de aplicar el operador en el nodo utilizando como operando los valores de los hijos.* (Allen, 2004: 148)

Entre las propiedades fundamentales de los árboles se pueden identificar:

- Un nodo es distinguido como la raíz
- Todo nodo C, excepto la raíz, está conectado por medio de una arista a un único nodo p, p es el padre de c, y c es uno de los hijos de p.
- Hay un único camino desde la raíz a cada nodo. El número de aristas que deben atravesarse es la longitud del camino.
- Los nodos que tienen el mismo padre son hermanos
- Los nodos que no tienen hijos son denominados hojas.
- Sólo puede haber un único nodo sin padres, que llamaremos raíz.
- Los demás nodos (tienen padre y uno o varios hijos) se les conoce como rama.

La raíz de un árbol se representa gráficamente en la parte superior, en este esquema está representada una relación jerárquica a partir del nodo raíz "en sentido vertical descendente, definiendo niveles. El nivel del nodo raíz es 1. Desde la raíz se puede llegar a cualquier nodo progresando por las ramas y atravesando los sucesivos niveles estableciendo así un camino." (OWC, 2012: 144)

Un nodo que referencia un nodo debajo suyo es un nodo padre, de forma similar, un nodo referenciado por un nodo encima de él, es un nodo hijo. Un nodo podría ser un

padre e hijo, o un nodo hijo y un nodo hoja. Un árbol con N nodos debe tener N-1 aristas porque a cada nodo, excepto a la raíz, le llega una arista.

La profundidad de un nodo en un árbol es la longitud del camino que va desde la raíz hasta ese nodo, es por ello que la profundidad de la raíz es siempre 0 y la de cualquier nodo es la de su padre más 1. Por otra parte la altura de un nodo consiste en la longitud del camino que va desde el nodo hasta la hoja más profunda bajo él. La altura de un árbol se define como la altura de su raíz.

> *Si hay un camino del nodo u al nodo v, entonces decimos que u es un ascendiente de v y que v es un descendiente de u. Si u≠v, entonces u es un ascendiente primero de v y v es un descendiente propio de u. El tamaño de un nodo es igual al número de descendientes que tiene (incluyendo dicho nodo). El tamaño de un nodo se define como el tamaño de su raíz.* (Allen, 2004: 436)

Se dice que un árbol es completo cuando todos sus nodos (excepto las hojas) tienen el mismo grado y los diferentes niveles están poblados por completo. A veces resulta necesario completar un árbol añadiéndole nodos especiales.

Las operaciones fundamentales de los árboles consisten en:

Buscar(k):devuelve un elemento con clave k en el conjunto o el valor nulo si no existe.

Insertar (x): Incluye un elemento x al conjunto.

Eliminar(k): Excluye el elemento con calve k en el conjunto.

Mínimo(): Devuelve el elemento que tiene el menor valor de clave, o el valor nulo si el conjunto es vacío.

Máximo(): Devuelve el elemento del conjunto que tiene el mayor valor de clave o nulo si el conjunto es vacío.

Los árboles binarios son definidos en “Estructuras de Datos” (OWC, 2012) como árboles de grado 2. Mientras que Peral (2009) ofrece una definición más amplia:

> *Un árbol binario es un conjunto de elementos del mismo tipo tal que: o bien es el conjunto vacío, en cuyo caso se denomina árbol vacío o nulo o bien no es vacío, y por tanto existe un elemento distinguido llamado raíz, y el resto de los elementos se distribuyen en dos subconjuntos disjuntos, cada uno de los cuales es un árbol binario llamados, respectivamente subárbol izquierdo y subárbol derecho del árbol original.*

Entre los árboles binarios se encuentran los llamados colas con prioridad, los cuales soportan acceso y eliminación del mínimo de una colección de elementos (ICT, 2012), mientras que los árboles binarios de búsqueda (ABB) permiten inserciones y acceso a los elementos en tiempo logarítmico.

Existen dos tipos de recorridos de los árboles binarios, uno de ellas es en amplitud: el cual implica un acceso a las claves recorriendo en cada nivel de izquierda a derecha. (OWC, 2012). Por otra parte está el recorrido en profundidad en el cual se avanza en forma vertical desde la raíz hacia las hojas y, en principio, de izquierda a derecha.

Existen tres variantes de los recorridos: el primero en preorden, donde la clave se procesa en primer lugar. "Posteriormente se realizan las llamadas recursivas (por la izquierda y por la derecha). Este tipo de recorrido es el más utilizado pues atiende a la estructura jerárquica del árbol. También es el más eficiente cuando se trata de buscar claves." (OWC, 2012:148). Una segunda opción consiste en el inorden: "Se desciende recursivamente por la rama izquierda. Al alcanzar la condición de finalización *(árbol= =null)* se retorna y se procesa la clave, y a continuación se progresa por la rama derecha. Este tipo de recorrido es especialmente útil en todos aquellos procesos en los que sea necesario recorrer los nodos de acuerdo al orden físico de los mismos." (Ídem) Y por último el postorden, donde "se desciende recursivamente por la rama izquierda, al alcanzar el final de dicha rama, se retorna y se desciende por la rama derecha. Cuando se alcanza el final de la rama derecha se procesa la clave. este recorrido es el menos utilizado. Se utiliza para liberar memoria, o bien cuando la información de los niveles más profundo del árbol afecta a la información buscada" (Ídem)

En *Estructuras de Datos en Java™ Compatible con Java™ 2* Volumen I se identifica como ABB a un árbol que permite la inserción, eliminación y búsqueda. "También se puede utilizar para acceder al *k*-ésimo menor elemento. El coste en tiempo es logarítmico en promedio para una implementación simple, y logarítmico en el caso peor, con una implementación más cuidada". (Allen, 2004: 149)

Pero en ese propio texto se plantea una definición con la cual coinciden la mayoría de los autores: "El árbol binario de búsqueda es un árbol binario que satisface la propiedad de la búsqueda ordenada. Esto significa que para cada nodo X del árbol, los valores de todas las claves de su subárbol izquierdo son menores que la clave de X y los valores de todas las claves de su subárbol derecho son mayores que la clave de X." (Ídem: 467)

El principal interés en cuanto a los ABB consiste en que su recorrido inorden proporciona los elementos ordenados de forma ascendente y en que la búsqueda de algún elemento suele ser muy eficiente. De hecho, "esta propiedad define un método de ordenación similar al Quicksort, con el nodo raíz jugando un papel similar al del elemento de partición del Quicksort aunque con los ABB hay un gasto extra de memoria mayor debido a los punteros". (De la Torre, 2012)

"Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno". (Gutiérrez, 2010)

La mayoría de las operaciones en los ABB son sencillas de comprender (Allen, 2004: 190) y toman un tiempo proporcional a la altura del árbol (González, 2012) Entre las operaciones fundamentales se encuentran:

- Insertar O(n) peor caso O(Log n) mejor caso
- Buscar O(n) peor caso O(Log n) mejor caso
- Eliminar O(n) peor caso O(Log n) mejor caso
- Buscar Min – BuscarMax O(n) pc O(Log n) mejor caso
- Buscar el K-esimo menor elemento

1.3.1 Búsqueda

La búsqueda en árboles binarios es un método de búsqueda simple, dinámico y eficiente considerado como uno de los fundamentales en Ciencia de la Computación (De la Torre, 2012). Para buscar un elemento k en un ABB partimos de la raíz y comparamos k con la clave situada en la raíz. "Si coinciden la búsqueda finaliza con éxito, si k<r es evidente que k, de estar presente, ha de ser un descendiente del hijo izquierdo de la raíz, y si es mayor será un descendiente del hijo derecho. (Ídem) Luego, el valor de retorno de una función de búsqueda en un ABB "puede ser un puntero al nodo encontrado, o NULL, si no se ha encontrado". (EDD, 2000)

Resulta importante señalar que la búsqueda en este tipo de árboles es muy eficiente, de hecho requiere O(log2n) operaciones en el caso medio, en un árbol construido a partir de n claves aleatorias, y en el peor caso una búsqueda en un ABB con n claves puede implicar revisar las n claves, o sea, es O(n).

1.3.2 Inserción

El procedimiento de inserción es muy simple y tiene muchos puntos en común con el de buscar. “Se trata de crear un nuevo nodo en la posición que le corresponda según el criterio de árbol binario de búsqueda.” (OWC, 2012:149) Para insertar un elemento nos basamos en el algoritmo de búsqueda. “Si el elemento está en el árbol no lo insertaremos”. (EDD, 2000)

En el transcurso de la operación es muy importante no romper la estructura ni el orden del árbol, sobretodo no violentar la propiedad de los árboles binarios de que no pueden tener más de dos hijos. Por ello

> *si un nodo ya tiene 2 hijos, el nuevo nodo nunca se podrá insertar como su hijo. Para localizar el lugar adecuado del árbol donde insertar el nuevo nodo se realizan comparaciones entre los nodos del árbol y el elemento a insertar. El primer nodo que se compara es la raíz, si el nuevo nodo es menor que la raíz, la búsqueda prosigue por el nodo izquierdo de éste. Si el nuevo nodo fuese mayor, la búsqueda seguiría por el hijo derecho de la raíz. Este procedimiento es recursivo, y su condición de parada es llegar a un nodo que no tenga hijo en la rama por la que la búsqueda debería seguir. En este caso el nuevo nodo se inserta en ese hueco, como su nuevo hijo.* (Sama, 2012)

1.3.3 Eliminación

La operación más complicada es eliminar. El problema es que la eliminación de un nodo puede desconectar varias partes del árbol. Debemos ser cuidadosos en su manipulación de modo que se mantenga siempre la propiedad de la búsqueda ordenada. Además, es necesario evitar que el árbol tenga demasiada profundidad, puesto que (…) la profundidad del árbol influye en el tiempo de ejecución de los algoritmos. (Allen, 2004, 468)

Para borrar un elemento existen diferentes posibilidades. Primero, si se trata de un nodo hoja accedemos a borrarlo, este “es el caso más sencillo, únicamente habrá que borrar el elemento y ya habremos concluido la operación” (Sama, 2012). Otra opción consiste en que el elemento esté en un nodo interior, de esta forma eliminándolo, podríamos desconectar el árbol. “Para evitar que esto suceda se sigue el siguiente procedimiento: si el nodo a borrar (u) tiene sólo un hijo se sustituye “u” por ese hijo” (De la Torre, 2012). Por último el caso más complejo: cuando el nodo a eliminar tiene dos hijos, en este caso “se encuentra el menor elemento de los descendientes del hijo derecho(o el mayor de los

descendientes del hijo izquierdo) y se coloca en lugar de “u”, de forma que se continúe manteniendo la propiedad de ABB” (Ídem).

“El árbol binario de búsqueda soporta de modo eficiente las operaciones buscarMin y buscarMax. Para ejecutar buscar Min, partimos de la raíz y recorremos repetidamente todos los nodos izquierdos hasta llegar a un nodo que no tenga hijo izquierdo. Dicha hoja es el elemento mínimo del árbol. buscarMax es análoga, salvo por el hecho de que el recorrido es ahora por la derecha. Nótese que el coste de ambas operaciones es proporcional al número de nodos del camino de búsqueda. El coste en media es logarítmico pero puede ser lineal en el peor de los casos”. (Allen, 2004: 468)

Según Allen (2004) Tomo II, se puede concluir que el tiempo medio de ejecución de todas las operaciones es O(log N). Esto funciona en la práctica, pero no ha sido establecido analíticamente debido a que en la demostración de los resultados anteriores no se ha tenido en cuenta el efecto de las operaciones de eliminación.

Hay muchos tipos de árboles binarios de búsqueda. Los árboles AVL y los árboles rojo-negros son ambos formas de árboles binarios de búsqueda auto balanceados. Un árbol biselado es un árbol binario de búsqueda que automáticamente mueve los elementos a los que se accede frecuentemente cerca de la raíz. En los montículos, cada nodo también mantiene una prioridad y un nodo padre tiene mayor prioridad que su hijo.

1.4 AVL

Una publicación soviética del año 1962 propuso los árboles AVL, llamados así en honor de sus creadores Adelson-Velskii y Landis, fueron los primeros árboles binarios de búsqueda bien equilibrados, aunque, al igual que los rojo-negros, no están perfectamente equilibrados. Mark Allen, (2004) los define como:

“Se trata de árboles binarios de búsqueda con una condición adicional de equilibrio. Dicha condición debe ser fácil de mantener, asegurando que la profundidad del árbol sea siempre O(log N). La idea más sencilla es exigir que los subárboles izquierdo y derecho tengan la misma profundidad. La recursión nos indica que dicha idea se aplica a todos los nodos del árbol, ya que cada uno de ellos es a su vez la raíz de algún subárbol”. (Allen, 2004: 484-485)

De manera más simplificada Gurin (2004) propone un concepto con el cual coinciden exactamente todos los autores y que resulta reproducido en la mayoría de las conferencias sobre el tema: “Un árbol AVL es un árbol binario de búsqueda que cumple con la condición de que la diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho 1”.

La condición de equilibrio AVL implica que el árbol tiene siempre una profundidad logarítmica. Para demostrarlo, basta mostrar que un árbol de altura H debe tener, por lo menos, C^H nodos, donde C es una constante mayor que 1. En otras palabras, el número mínimo de nodos de un árbol crece exponencialmente con la altura. Así la profundidad máxima de un árbol con N elementos es Log N. (Allen, 2004: 486)

La propiedad que define un AVL es el equilibrio, llamdo también factor de balance. Al valor | Altura(Hijo_izq(a)) - Altura(Hijo_dch(a)) | lo podemos llamar factor de balance. (Gálvez, 2012), aunque (…) es complicado mantener la condición de equilibrio al insertar nuevos elementos. (Allen, 2004: 485) Esta misma propiedad asegura que la profundidad del árbol sea O(log(n)).

Si T es un árbol binario no vacío con TL y TR como subárboles izquierdo y derecho respectivamente, entonces T está balanceado con respecto a la altura si y solo si TL y TR son balanceados respecto a la altura, y | hl - hr | = 1 donde hl y hr son las alturas respectivas de TL y TR. (Canales, 2012) Para cualquier nodo T en un árbol AVL se cumple que el Factor de Equilibrio (FE)= -1, 0, 1. Si llegara a tomar los valores -2 o 2 debe restructurarse el árbol.

“Este factor indica si el árbol es *de izquierda pesada* (la altura del sub-árbol izquierdo es uno mayor que el sub-árbol derecho), *equilibrada* (dos subárboles tienen la misma altura) o hacia la *derecha pesada* (la altura del sub-árbol derecho es una mayor que el sub-árbol izquierdo)”(Morris, 1998).

Cada nodo, además de la información que se pretende almacenar, debe tener los punteros a los árboles derecho e izquierdo, es importante aclarar que no se trata de un equilibrio perfecto, pero sí suficiente como para que su comportamiento sea lo bastante bueno como para usarlos donde los ABB no garantizan tiempos de búsqueda óptimos. (Pozo, 2002) “El algoritmo para mantener un árbol AVL equilibrado se basa en requilibrados locales, de modo que no es necesario explorar todo el árbol después de cada inserción o borrado”. (Pozo, 2002)

Según Gálvez el secreto para mantener el factor de balance en los límites establecidos se encuentra tanto en el proceso de inserción como en el de eliminación, de gran importancia serán las rotaciones a derecha y a izquierda. (Gálvez, 2012).

Los árboles AVL pertenecen al grupo de los ABB, por lo que las operaciones son iguales. "Las nuevas operaciones son las de equilibrar el árbol, pero eso se hace como parte de las operaciones de insertado y borrado". (Pozo, 2002) Nuevamente las principales operaciones serían: buscar, insertar y eliminar.

"Implementaremos la inserción de elementos en un árbol AVL de forma análoga a como lo haríamos para árboles binarios de búsqueda salvo que en cada recursión del algoritmo verificaremos y corregiremos el equilibrio del árbol" (Gurin, 2004). Es decir, que luego de la operación debemos comprobar si la relación entre las alturas de los subárboles no ha cambiado, (que no cambie el Factor de Equilibrio), de ser así debemos entonces realizar la rotación correspondiente.

También es importante ir actualizando las alturas de cada nodo en cada recursión dado que las rotaciones, inserciones y eliminaciones pueden modificarlas (Gurin, 2004). Al decir de Allen, (2004: 492) "es más eficiente almacenar directamente en el nodo resultado de la comparación en lugar de guardar su altura. Esto evita cálculos repetitivos acerca del equilibrio. Por otra parte, en esta ocasión la recursión implica una sobrecarga substancial respecto a la versión iterativa. Esto es debido a que, hemos de recorrer el árbol hacia abajo para después volver hasta la raíz, en lugar de detenernos tan pronto como la rotación se realice. Como consecuencia, en la práctica, se emplean otros esquemas de equilibrio de árboles."

La estrategia para diseñar el algoritmo de eliminación sobre árboles AVL es la misma que para la inserción. Se utiliza el mismo algoritmo que sobre árboles binarios de búsqueda, pero "en cada recursión se detectan y corrijen errores por medio de balancear() y se actualiza la altura del nodo actual." (Gurin, 2004)

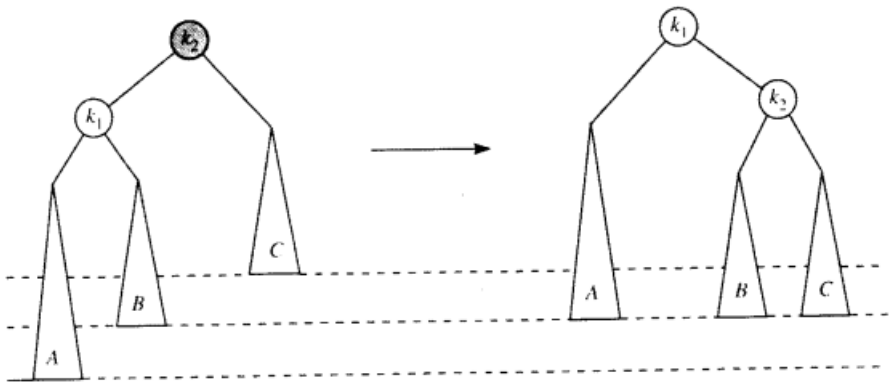
(…) Al ir recorriendo el camino hacia la raíz y actualizando la información de equilibrio, podemos encontrar un nodo cuyo nuevo equilibrio incumpla la condición AVL. (Allen, 2004: 486-487). Es por ello que en el caso de los AVL se emplean las rotaciones para restablecer la condición de equilibrio.

Cuando insertamos o eliminamos un nodo y se produce un desequilibrio en un nodo que incumple la propiedad de los árboles AVL y debemos resolver dicha sobrecarga con una rotación. Existen 4 tipos de rotaciones:

- A la izquierda del subárbol izquierdo.
- A la derecha del subárbol derecho.
- A la izquierda del subárbol derecho.
- A la derecha del subárbol izquierdo.

Los dos primeros casos se resuelven de forma análoga al igual que los dos últimos por lo que se pudiera decir que solo son 2 casos aunque desde el punto de vista computacional hay que seguir mencionando 4 casos. A los dos primeros casos se les denomina Rotación simple y a los otros Rotación doble.

En el primer caso donde el desequilibrio se produce por la izquierda



Fuente: Estructuras de Datos y Algoritmos en Java™

En el ejemplo, llamamos k_2 al nodo que incumple la propiedad de AVL, K_1 al hijo de k_2 que se encuentra en el camino desbalanceado, y por A, B y C a los subárboles izquierdo y derecho de K_1 así como el derecho de k_2 respectivamente. Como se ve en el ejemplo el subárbol A está dos niveles más profundo que el subárbol C, para poder requilibrar de forma correcta el árbol basta con hacer rotar los nodos K_1 y k_2 manteniendo los subárboles A y C con sus respectivos padres y permutando el subárbol B de hijo derecho de K_1 a hijo izquierdo de k_2 con lo cual se cumple la propiedad de que k_2 es mayor que K_1 y que B se encuentra entre K_1 y k_2. En otras palabras una rotación simple intercambia los papeles de los padres y de los hijos manteniendo la ordenación del árbol. El caso 4 se resuelve de igual forma a este.

Las rotaciones simples no funcionan en todas las ocasiones, puesto que pueden existir desequilibrios en "zig-zag, es decir desequilibrios que no son ni a la derecha ni a la izquierda. (…)En estos casos se aplica otro tipo de rotación denominado *Rotación doble* la cual, análogamente a la rotación simple, puede ser izquierda o derecha según el caso. En realidad, la rotación doble constará de dos rotaciones simples" (Gurin, 2004). Lo más importante de estas rotaciones es que permiten que se mantenga la condición AVL del árbol.

Un estudio sobre la complejidad de los algoritmos de inserción y eliminación realizado por Gálvez (2012), llega a la conclusión de que: "Al realizar una inserción de una sola clave se puede producir como máximo una rotación (de dos o tres nodos) El borrado puede requerir una rotación en todos los nodos del camino de búsqueda. Los análisis empíricos dan como resultado que, mientras se presenta una rotación por cada dos inserciones, sólo se necesita una por cada cinco borrados. El borrado en árboles equilibrados es, pues, tan sencillo (o tan complicado) como la inserción"

La altura de cada árbol determina la velocidad de las operaciones, cada vez que se realiza una operación es necesario calcular y recalcular las alturas, por lo que la complejidad superior al orden O (log n). En la especificación no constituye un problema, sin embargo en las implementaciones supone un recargo de tiempo considerable. Se dice que si el árbol crece o decrece descontroladamente, el rendimiento puede disminuir considerablemente, siendo para el caso más desfavorable (insertar un conjunto de claves ordenadas en forma ascendente o descendente) el coste de acceso: O(N).

Las aplicaciones de estos árboles coinciden con las de los árboles binarios de búsqueda, aunque pueden emplearse además en sistemas en tiempo real en los que se necesita establecer una cota superior aceptable del tiempo que tardarán en ejecutarse algunas operaciones.

1.5 Árboles Rojinegros

Los árboles rojo-negros, rojinegros o coloreados (RN), resultan una alternativa popular a los AVL. Su estructura original fu creada por Rudolf Bayer en 1972, nombrándolos "árboles-B binarios simétricos", pero no fue hasta un trabajo de Leo J. Guibas y Robert Sedgewick de 1978, que adquieren su nombre moderno. En relación con los AVL, "una

cuidadosa implementación no recursiva de los árboles RN es más simple y eficiente" (Allen, 2004: 492).

Cada nodo contiene un dato que indicará su color, de hecho existen reglas para asignar el color a cada nodo, por lo que "una trayectoria cualquiera desde un nodo a una hoja no sea mayor que el doble del largo de cualquier otra" (Silva, 2008). Así se asegura el balance del árbol, así como el costo logarítmico para las operaciones en peor caso. Estos árboles pueden buscar, insertar y borrar en un tiempo O(log n), donde n es el número de elementos del árbol.

Existen cuatro propiedades de obligado cumplimiento para los árboles RN, aunque algunos autores reconocen hasta seis propiedades esenciales, Allen (2004) identifica a cuatro de ellas como imprescindibles:

1. Cada nodo está coloreado con los colores rojo o negro

2. La raíz es negra. (Allen, 2004) Los nodos externos, son negros; éstos no son nodos con información. (Silva, 2008)

3. Si un nodo es rojo, sus hijos deben ser negros.

4. Todos los caminos desde un nodo a una referencia null deben contener el mismo número de nodos negros." (Allen, 2004: 492-493)

La altura de un árbol RN con n nodos internos es, a lo máximo $h \leq 2\log(n + 1)$. La función altura negra de un nodo x bh(x), se define como el número de nodos negros de cualquier trayectoria desde x hasta una hoja, sin contar x. Además se puede demostrar por inducción que si cualquier camino desde la raíz a una referencia null contiene B nodos negros, entonces "el árbol debe tener, al menos $2^B - 1$ nodos negros. Más aún, como la raíz es negra y no pueden existir dos nodos consecutivos de color rojo en un camino, la altura de un árbol RN es a lo sumo 2log(N+1). Como consecuencia, está garantizado que la búsqueda es una operación logarítmica" (Allen, 2004: 492-493).

En los árboles rojo-negros las hojas no son relevantes y no contienen datos. A la hora de implementarlo en un lenguaje de programación, para ahorrar memoria, un único nodo (nodo-centinela) hace de nodo hoja para todas las ramas. Así, todas las referencias de los nodos internos a las hojas van a parar al nodo centinela.

Una de las desventajas de los árboles RN, es que luego de la inserción o eliminación puede perderse alguna de sus propiedades elementales. "Una vez detectada una

violación, se deben tomar medidas que reparen el balance del árbol. Estas medidas utilizan dos herramientas: rotaciones y cambios de color" (ICT, 2012), aunque se debe tener en cuenta que:

- Cambiar un nodo de rojo a negro no afecta la condición 3, pero afecta la condición 4 (altura negra se incrementa en todos los nodos ascendientes)
- Cambiar un nodo de negro a rojo puede afectar la condición 3(si el padre o alguno de los hijos es rojo) y también la condición 4 (altura negra se incrementa en todos los nodos ascendientes)
- Si como resultado de una operación la raíz pasa a ser rojo, se puede cambiar a negro directamente sin afectar las condiciones.
- Borrar un nodo rojo no afecta las condiciones, pero borrar un nodo negro sí (altura negra decrece en los descendientes. (UVA, 2012)

Operaciones Fundamentales:

- *Eliminación*

  El número de nodos consultados en una búsqueda de un árbol RN medio es casi idéntico al número de nodos consultados en los árboles AVL, "aunque estáticamente las probabilidades de equilibrio de los árboles RN son ligeramente más débiles"(Allen, 2004: 496-497). La ventaja de los árboles RN es la sobrecarga relativamente pequeña que se necesita para realizar una inserción y el hecho de que, en la práctica, las rotaciones son bastante pequeñas.

- *Inserción (ascendente y descendente)*

  En el caso de la operación de inserción las alturas negras del nodo denominado abuelo deben ser iguales antes y después de la inserción; se debe partir de que el nodo a insertar será en principio rojo, ya que de otra forma incumpliría la propiedad 4. "Si se inserta un nodo con color rojo, y si el árbol estaba vacío, debe cambiarse a negro para mantener la propiedad de que la raíz sea negra. Si se inserta un nodo en la raíz, como ésta es negra, se mantienen las propiedades de los árboles coloreados. (Silva, 2008)

  Si el padre fuera negro (como en el ejemplo anterior) ya hemos acabado. En caso contrario se violaría entonces la tercera propiedad, por lo que el árbol debe ser

ajustado, cuidando de no incumplir la propiedad 4. Existen entonces varios casos a considerar cuando el padre es rojo (cada uno con su correspondiente simétrico).

Primero: el hermano del padre es negro, "adoptamos el acuerdo de que las referencias null son negras" (Allen, 2004: 493). Sea X la nueva hoja añadida, P su padre, S el hermano del padre (si existe) y G el abuelo. En este caso solo X y P son rojos, G es obligatoriamente negro, para no incumplir la propiedad 3. Respecto a G, X puede ser un nodo interior o exterior. "Si X es un nieto exterior, la propiedad 3 se recupera mediante una rotación simple entre su padre y su abuelo, con algunos cambios de color. Si X es un nieto interior, es suficiente unja rotación doble y algunos cambios de color." (Allen, 2004: 494)

Cuando el hermano del padre es rojo (Ver Anexos), no funciona ni la rotación simple ni la doble luego de una inserción, ya que sus resultados incluyen dos nodos rojos consecutivos; de hecho es fácil comprobar que existen tres nodos en los caminos hasta los hijos de S y solo uno de ellos puede ser negro. "Esto nos indica que tanto S como la nueva raíz del subárbol deben ser rojas" (Allen, 2004: 495), así una rotación simple entre el padre y el abuelo, así como los necesarios cambios de color, pueden restablecer la propiedad 3. En el caso de que el padre de la raíz del subárbol (bisabuelo inicial de X) también fuera rojo, podemos filtrar este mecanismo hacia atrás hasta que, o bien se llegue a la raíz (que será negra) o ya no se encuentren nodos rojos consecutivos, claro que "entonces deberíamos realizar un recorrido ascendente hacia la raíz, al igual que los árboles AVL" (Allen, 2004: 495).

Un caso particular es cuando el tío no existe, es decir cuando es un nodo externo; en este caso se considera tío de color negro.

Para evitar la eventual necesidad de tener que hacer un doble recorrido del árbol. aplicamos un procedimeinto descendiente mientras se busca el punto de inserción. Más concretamente, "se garantiza que cuando llegamos a una hoja e insertamos un nodo S no será rojo"(Allen, 2004). Entonces basta con añadir una hoja roja y, si es necesario emplear una sola rotación (simple o doble). El proceso es conceptualmente sencillo.

En el camino descendente, si un nodo X (la nueva hoja añadida) tiene dos hijos rojos, el color de X cambia a rojo y el de sus dos hijos a negro. El número de nodos negros en los caminos por debajo de X permanece inalterable. Si X es la raíz, la

convertiríamos en roja, hay que volver a negro (esto no puede violar ninguna de las reglas). Si el padre de X es rojo, hay que aplicar rotación simple o doble. "¿Qué pasa si el hermano del padre de X es también rojo? Esta situación NO puede darse, gracias al proceso efectuado en el camino descendiente" (ICT, 2012).

- *Eliminación descendente*

El algoritmo usual de eliminación en los árboles binarios de búsqueda elimina nodos que son hojas o tienen un único hijo. "(…) nunca se borran nodos con dos hijos: solo se remplaza su contenido" (Allen, 2004: 503). Si el nodo que vamos a borrar es rojo no hay ningún problema. Sin embargo, si es negro su eliminación hará que se incumpla la cuarta propiedad. "La solución al problema consiste en asegurar que cualquier nodo que eliminemos será rojo" (Allen, 2004: 503).

Mientras que *Estructuras de Datos en Java™ Compatible con Java™ 2* explica la operación eliminar a partir de la combinación de colores del nodo a eliminar, su padre y su hermano, Silva explica que deben analizarse tres casos, cuando tiene dos descendientes, uno solo o ninguno.

Otra explicación, a la cual se adhiere la presente investigación es la siguiente, la cual plantea que la operación de borrado se lleva a cabo de la misma manera que en los ABB:

- "Se busca el nodo a borrar (como los nodos hojas son nodos nulos, en caso de existir el nodo a borrar debe tener siempre dos hijos.
- Si es un nodo con dos hijos no nulos, se busca el mayor nodo (el más a la derecha) de su subárbol izquierdo (lo llamamos x), se intercambia sus datos con el nodo a borrar y se pasa a borrar el nodo x. Como es el nodo más a la derecha, su hijo derecho será nulo." (UVA, 2012)

Lo importante de la operación es el proceso de ajuste o restructuración que tendrá lugar luego de la eliminación, en este caso llamaremos P al nodo que se borra, x es el hijo nulo, o uno cualquiera de los hijos si ambos son nulos, z es el padre del nodo borrado e y es el nodo hermano del nodo borrado. Es imprescindible conocer los nodos x y z, donde el primero puede ser nulo, pero z debe existir.

Este proceso consiste en una comprobación de casos triviales, en caso de no ser ninguno de ellos se entra en un bucle, donde en cada iteración x y z apuntan a nodos

con la siguiente estructura: “los nodos z e y no son nulos y el nodo x puede serlo” (UVA, 2012) Se comprueba entonces en cuál de los cinco casos no triviales estamos, se efectúa la operación adecuada y se decide si se continúa con la siguiente iteración. (En lo que sigue se considera que x es un hijo izquierdo. Existen otros casos no triviales cuando x sea un hijo derecho, que son totalmente simétricos respecto a los explicados). A continuación se explican todos los posibles casos a encontrar durante una operación de reajuste:

Casos triviales:

a) Nodo borrado es rojo: El árbol sigue siendo rojo-negro, por lo que no hay que realizar ningún ajuste.

En lo que sigue consideramos que le nodo borrado es negro.

b) El hijo, x, es rojo: En este caso se incumple la condición de que los hijos de z tengan la misma altura negra. Cambiando el color de x a negro se restablece la condición.

En lo que sigue consideraremos que el nodo borrado y su hijo son negros. Para destacar los casos es necesario fijarse en el color del nuevo padre (z) y, sobre todo, en el del hermano (y). En un caso incluso hay que tener en cuenta el color de los sobrinos.)

Se debe tener en cuenta que tanto x como y pueden ser nulos (serían nodos negros) y que si un nodo es rojo entonces obligatoriamente no es nulo y por lo tanto tiene hijos. (UVA, 2012)

Casos imposibles:

Hermano negro nulo: Si el hermano de x es negro y nulo tiene altura negra 1, por lo tanto z debería tener (tras el borrado) una altura una unidad menor, es decir 0. Pero z es un nodo negro, y aunque sea nulo tendría altura 1. Por lo tanto, no se puede dar que x sea negro y esté desequilibrado respecto a un hermano nulo: el hermano debe existir. (UVA, 2012)

Rotaciones

Caso 1: Hermano rojo, padre negro

Solución: Se hace una rotación padre-hermano y se cambian los colores. El nodo x pasa a tener como nuevo hermano al nodo a (hijo derecho de y), sigue teniendo

una altura menor en uno que la de su hermano, pero ahora su padre es rojo y por lo tanto caerá en alguno de los siguientes casos (dependiendo del color de a)

Iteración siguiente: Se vuelve a comprobar con los mismo nodos x y z, se garantiza que no se cae en este caso. (UVA, 2012)

Caso 2 (hermano negro, no nulo, sobrinos negros, padre negro)

Solución: Se cambia el color del hermano (y) a rojo. Con ello los nodos x e y pasan a tener la misma altura negra. El problema es que la altura de z ha disminuido.

Iteración siguiente: se vuelve a comprobar, ahora el nodo llamado x es el nodo z y el nodo llamado z es el padre de z (si z es la raíz el árbol cumple todas las condiciones y se termina el bucle) (UVA, 2012)

Caso 3 (hermano negro no nulo, sobrinos negros, padre rojo)

Solución: Se cambia el color del hermano (y) a rojo y el del padre (z) a negro. Con ello los nodos x e y pasan a tener la misma altura negra. La altura de z sigue siendo la misma.

Iteración siguiente: El árbol ya cumple todas las condiciones. Se termina el bucle. (UVA, 2012)

Caso 4 (hermano negro no nulo, sobrinos rojo/negro, padre cualquier color)

Solución: Se realiza una rotación hermano-sobrino izquierdo y se cambian sus colores. Los hijos del sobrino izquierdo existen 8aunque pueden ser nulos) y son negros, ya que el sobrino izquierdo es rojo. El nodo x pasa a tener como hermano al nodo a y sigue teniendo una altura negra menor en uno que la de su hermano.

Iteración siguiente: Se vuelve a comprobar con los mismos nodos x y z, se caerá en el caso 5 ya que el hermano sigue siendo negro y los sobrinos son negro y rojo. (UVA, 2012)

Caso 5 (hermano negro no nulo, sobrinos cualquiera/rojo, padre cualquier color)

Solución: se realiza una rotación padre-hermano y se cambia el color de la siguiente forma: El padre (z) pasa a ser negro, el hermano (y) toma el mismo color que el que tenía originalmente z, el sobrino derecho pasa de rojo a negro (ese sobrino debía existir ya que era rojo)

Iteración siguiente: El árbol ya cumple todas las condiciones. Se termina el ajuste. (Ídem)

Nota: en ningún caso se cambia el color de x, por lo que puede perfectamente ser un nodo nulo. (Ídem)

No es solamente los tiempos de las operaciones lo que hacen valiosos a los árboles rojinegros en aplicaciones sensibles al tiempo, como las de tiempo real; son apreciados además para la construcción de bloques en otras estructuras de datos que garantizan un peor caso, como por ejemplo muchas de las utilizadas en geometría computacional. Son importantes también en programación funcional, donde resultan una de las estructuras de datos persistentes más comúnmente utilizadas en la construcción de arreglos asociativos y conjuntos que pueden retener versiones previas tras mutaciones.

1.6 Árboles B

La necesidad de mantener índices de almacenamiento externo para acceso a bases de datos, (teniendo en cuenta la lentitud de estos dispositivos) generó el interés de aprovechar su capacidad de almacenamiento para una cantidad de información altamente organizada, de manera que el acceso a una clave sea lo más rápido posible. Una solución pueden ser los árboles M-arios, los cuales se definen de una forma similar a los ABB

"En un árbol de búsqueda M-ario, son necesarias M-1 claves para decidir cuál de las ramas debemos utilizar. Para conseguir que este esquema sea eficiente en el peor de los casos, es preciso asegurarnos que el árbol M-ario esté equilibrado de alguna forma. (…) tampoco podemos permitirnos que nuestro árbol M-ario degenere en un árbol binario, ya que en tal caso volveríamos a los log N accesos." (Allen, 2004)

El objetivo principal de estos árboles consiste en que la altura sea pequeña, pues "las iteraciones y los acceso a disco dependerá de ello." (Gálvez, 2012) Una forma de implementar todos los requisitos consiste en el empleo de los B-árboles de los cuales se conocen muchas variaciones y mejoras (Allen, 2004),entre ellas los B* y los B+, por lo que su implementación no carece de dificultad. Sin embargo, es sencillo comprobar que, en principio, el uso de un B-árbol garantiza tener que realizar pocos accesos a disco. (Allen, 2004)

Sus creadores Rudolf Bayer y Ed McCreight, no explicaron la utilización de la letra B de su nombre, se cree que proviene de su propiedad de balanceados, aunque también

puede referirse a Bayer, o a Boeing, porque sus creadores trabajaban en los Boeing ScientificResearchLabs por ese entonces. Lo que sí es seguro es que la B no proviene de binarios, ya que los B-trees (en inglés) nunca son binarios.

"Un árbol B de orden p es básicamente un árbol de búsqueda n-ario donde los nodos tienen p hijos como máximo, y en el cual se añade la condición de balanceo de que todas las hojas estén al mismo nivel". (Marín, 2012: 169) Entre otras definiciones también está la de Schaeffer (2012): "Árboles B son árboles balanceados que no son binarios. Todos los vértices contienen datos y el número por datos por vértice puede ser mayor a uno".

Allen (2004), en su Volumen III resume las propiedades que identifican toda la bibliografía consultado sobre el tema:

1. Los datos se almacenan en las hojas.
2. Los nodos internos contienen M-1 claves para guiar la búsqueda: la clave i representa la menor clave en el subárbol i+1.
3. La raíz es una hoja o tiene entre 2 y M hijos.
4. Todos los nodos internos, excepto la raíz, tienen entre (M/2) y M hijos y entre (M/2 - 1) y (M-1) elementos.
5. Todas las hojas se encuentran a la misma profundidad y tienen entre (L/2) y L hijos para un cierto valor fijo L. (Allen, 2004: 515)

Los nodos internos de los árboles-B deben tener un número variable de nodos hijos dentro de un rango predefinido. Cuando se inserta o elimina un dato de la estructura, varía la cantidad de nodos hijo dentro de un nodo, para que se mantenga el número dentro del rango predefinido, los nodos interno se juntan o se parten. Otra de sus propiedades consiste en que el conjunto de claves que se sitúan en un nodo cumplen con la condición: "k1< k2<…<km-1. De forma que los elementos que cuelgan del primer hijo tienen una clave con valor menor que K1 , los que cuelgan del segundo tienen una clave con valor mayor que K1 y menor que K 2, etc." (Araya, 2012)

- *Inserción*

  El esquema del algoritmo de inserción de una clave x en un árbol B de orden p sería el siguiente:

  1. "Buscar el nodo hoja donde se debería colocar x. (…) Si el elemento ya está en el ´árbol, no se vuelve a insertar" (Marín, 2012: 172)
  2. Si la clave no se encuentra en el árbol habremos llegado a una hoja, "que es justamente el lugar donde debemos realizar esa inserción.

Situados en un nodo donde realizar la inserción si no está completo, es decir, si el número de claves que existen es menor que el orden menos 1 del árbol, el elemento puede ser insertado y el algoritmo termina." (Araya, 2012)

3. En caso de que no quedaran sitios libres, "cogemos las p - 1 entradas de la hoja y x. La mediana m pasa al nodo padre, así como los punteros a sus nuevos hijos: los valores menores que m forman el nodo hijo de la izquierda y los valores mayores que m forman el nodo hijo de la derecha."(Marín, 2012: 172)

4. Si el nodo padre está completo, "continuamos la división de nodos hasta que encontremos un padre que no necesite ser dividido o bien la raíz del árbol (Allen, 2004) esta idea ya ha sido utilizada en los árboles RN. Si dividimos la raíz, obtenemos dos raíces, lo cual no es posible, por lo que generar una nueva raíz resulta una solución, cuyos hijos serían las dos raíces previamente obtenidas" (Allen, 2004). Es así como estos árboles ganan altura.

- *Eliminación:*

Mientras que la inserción se apoya en la división de los nodos, la eliminación se caracteriza por el proceso contrario: la unión de dos nodos en uno nuevo. Básicamente, esta operación consiste en buscar el elemento a borrar y después eliminarlo.

Dada una clave x a borrar, si se encuentra en un nodo interno, no se puede suprimir directamente. En tal caso "habría que sustituir x en el ´árbol por la siguiente clave en orden –es decir, la mayor del subárbol izquierdo o la menor del subárbol derecho de x– y se continúa con la eliminación como si se hubiera producido en la posición sustituida" (Marín, 2012: 173). En definitiva, la eliminación debe siempre producirse en una hoja.

Si la hoja en que se encuentra el elemento a borrar tuviese solo el mínimo número de elementos permitido, surge otro problema: luego de la operación se queda por debajo de este mínimo. "Podemos arreglar esta situación adoptando el hijo de un vecino, si es que el vecino no tiene también el mínimo número de hijos" (Allen, 2004). Este proceso es identificado en (Araya, 2012) como "redistribución", mientras que la "unión" se aplica en otra situación: cuando el vecino tiene el mínimo permitido.

En ese caso se puede combinar una hoja con dicho vecino para obtener una hoja completa, pero esto significa entonces que el padre ha perdido un hijo; si a la vez esta operación causa que el padre esté bajo mínimos reiteramos el mismo

procedimiento. Este proceso puede repetirse a lo largo del camino hacia la raíz (Allen, 2004). La raíz no puede tener un único hijo, así que si, como resultado de la adopción, se queda con un solo hijo, lo que haremos será eliminarle, convirtiendo a su hijo en la nueva raíz del árbol (Ídem). Esta es la única manera de que un B-árbol pierda altura.

- *Búsqueda*

  La búsqueda resulta la una operación simple, para empezar nos situamos en el nodo raíz del árbol, si la clave buscada se encuentra la operación ha terminado, si no "seleccionamos de entre los hijos el que se encuentra entre dos valores de clave que son menor y mayor que la buscada respectivamente y repetimos el proceso hasta que la encontremos" (Araya, 2012). En caso de que se llegue a una hoja y no podamos proseguir la búsqueda la clave no se encuentra en el árbol.

1.7 Otros tipos de árboles: quadtree y octree

Una gran variedad de estructuras existen para representar los datos espaciales. Una técnica normalmente usada es Quadtree. El desarrollo de éstos fue motivado por la necesidad de guardar datos que se insertan con valores idénticos o similares.

El término Quadtree se usa para describir una clase de estructuras jerárquicas cuya propiedad en común es el principio de recursividad de descomposición del espacio. Se emplea básicamente representar puntos, áreas, curvas, superficies y volúmenes. La descomposición puede hacerse en las mismas partes en cada nivelado (la descomposición regular), o puede depender de los datos de la entrada. La resolución de la descomposición, en otros términos, el número de tiempos en que el proceso de descomposición es aplicado, puede tratarse de antemano, o puede depender de las propiedades de los datos de la entrada.

Entre sus funciones fundamentales resalta la representación de un área bidimensional, por lo que resulta este el tipo más estudiando de los quadtree. Este ejemplo es basado en la subdivisión sucesiva del espacio en cuatro cuadrantes del mismo tamaño. Existen subcuadrantes con datos (Área Negra), vacíos (Área blanca) y combinados (Área ceniza). Cada cuadrante representa un nodo del Quadtree, los espacios negros y blancos siempre están en las hojas, mientras todos los nodos interiores representan los espacios grises.

Los árboles Quadtrees se puede clasificar según el tipo de datos que representan, incluyendo áreas, puntos, líneas y curvas. Quadtrees puede también ser clasificado independientemente de la forma que tenga por la información que contiene. Estas estructuras son las análogas bidimensioanles de los árboles octrees.

Se puede definir las octree como estructuras de datos donde cada nodo interno tiene exactamente 8 hijos. Estas estructuras son empleadas generalmente para particionar un espacio tridimensional, al dividirlo recursivamente en ocho octantes.

Los árboles quadtree dividen alrededor de un punto. En cuanto a una región punto (PR) octree, el nodo almacena un punto tridimensional explícito, el cual es el "centro" de la subdivisión para ese nodo; el punto que define una de las esquinas para cada uno de los ocho hijos. En una octree MX, el punto de subdivisión es implícitamente el centro del espacio que el nodo representa. El nodo raíz de una PR octree puede representar un espacio infinito; el nodo raíz de una octree MX debe representar un espacio con límite finito para que los centros implícitos estén bien definidos.

Entre sus principales aplicaciones se encuentran: indice espacial, detección de colisiones eficiente en tres dimensiones, determinación de cara oculta, método multipolo rápido, métodos no estructurados, método de los elementos finitos, octree de vóxeles escasos y teorema de Bayes[1]

Entre los ejemplos más sobresalientes se encuentra la el algoritmo para la cuantización del color, donde hay tres componentes de colores en el sistema modelo de color RGB (del inglés Red, Green, Blue; "rojo, verde, azul"). En índice nodo para ramificar desde en el nivel tope está determinado por una fórmula que usa los bits más significativos de los componentes de colores rojo, verde, y azul. el siguiente nivel bajo usa el siguiente bit significativo, y etc. los bits menos significativos se ignoran en algunas ocasiones para reducir el tamaño del árbol.

Capítulo 2: Diseño e implementación de las variantes de Árboles Binario de Búsqueda (ABB)

En este capítulo se explica el diseño de las interfaces y las clases que se proponen en la solución del problema, el cual será resuelto a partir de la definición de una biblioteca en forma de paquete para estudiar, modificar y usar las tres variantes de árboles binarios de búsquedas implementadas y de la aplicación de escritorio que muestra de forma gráfica las operaciones básicas de los tres tipo de árboles binarios de búsqueda implementados.

2.1 Requerimientos de la solución

En el programa de la asignatura Estructura de Datos I se pide entre las habilidades a desarrollar por parte de los estudiantes "Utilizar Tipos de Datos ya existentes y en algunos casos, asumir la concepción y el diseño de nuevos Tipos de Datos siendo consecuente con los postulados fundamentales que caracterizan al paradigma de la programación orientada a objetos" (referencia al Plan de estudio) por lo que en la impartición de la asignatura se trabaja en que los estudiantes logren implementar las ED estudiadas, modificar algunas de las implementaciones de las operaciones que las caracterizan y usar esas estructuras de datos como parte de otras clases que solucionan problemas o directamente en la solución de un problema. Esto hace necesario que el estudiante sepa:

a) Implementar y modificar implementaciones de las estructuras de datos estudiadas.
b) Usar esas estructuras de datos estudiadas en la solución de problemas.

Teniendo en cuenta lo anterior se ha decidido como parte de la solución dada en el presente trabajo que el estudiante pueda, siguiendo las orientaciones del profesor:

a) Acceder al código fuente de las implementaciones, el cual se ofrece con comentarios de Java (referencia a Java) que explican el algoritmo implementado y con comentarios de documentación para generarla usando javadoc (referencia a javadoc)
b) Utilizar paquetes de java (referencia a java) con las implementaciones de los cuatro tipos de árboles en el desarrollo de aplicaciones.

Adicionalmente y como parte del apoyo al proceso de aprendizaje de los algoritmos de las operaciones fundamentales de 3 de estos tipos de árboles (ABB, AVL y RN), se ha

desarrollado una aplicación de escritorio que en un ambiente visual agradable permite que el usuario realice las operaciones básicas sobre árboles y obtenga una representación visual de lo que sucede con el árbol durante la ejecución de la operación. Este software resulta de gran interés debido a que las operaciones de inserción y eliminación de algunas de las variantes de árboles son complicadas y se analizan varios casos los cuales son necesarios mostrar a los estudiantes para una mejor comprensión de cómo debe realizarse la implementación computacional de dicha operación.

2.2 Diagrama de clases del paquete Tree.

El paquete Tree está conformado por 4 sub-paquetes internos que describen como se implementan los árboles Binarios de Búsqueda y sus variantes más avanzadas (AVL y ARN).

En general, la concepción del diseño realizado agrupa las interfaces definidas y las clases abstractas en un sub-paquete, a partir del cual se definen tres sub-paquetes más, uno para cada implementación concreta de los tipos de árboles estudiados.

La representación del árbol utilizada es la de lista doblemente enlazada, por lo que cada nodo del árbol se representa por una llave y dos referencias al hijo izquierdo y al hijo derecho del nodo.

Un primer paquete llamado Tree.Abstract, contiene las dos interfaces utilizadas en la implementación: IBinarySearchNode e IBinarySearchTree, esta última para declarar las tres principales operaciones de estos tipos de árboles (insertar, eliminar y buscar). Como se puede observar en la figura 2.1, el paquete se completa con las clases abstractas AbstractBinarySearchNode y AbstractBinarySearchTree que implementan las interfaces anteriores y declaran los atributos y métodos generales que van a heredar cada una de las clases que representan nuestros distintos tipos de árboles.

Vale destacar que en la clase abstracta que representa al árbol binario de búsqueda además, de los métodos correspondientes a las tres operaciones fundamentales, se declaran varios métodos protegidos que apoyan la implementación de los métodos definidos en la interfaz. Se definen también varios métodos abstractos que responden a las necesidades comunes de estos tipos de árboles y que deben ser implementados adecuadamente en cada uno de ellos.

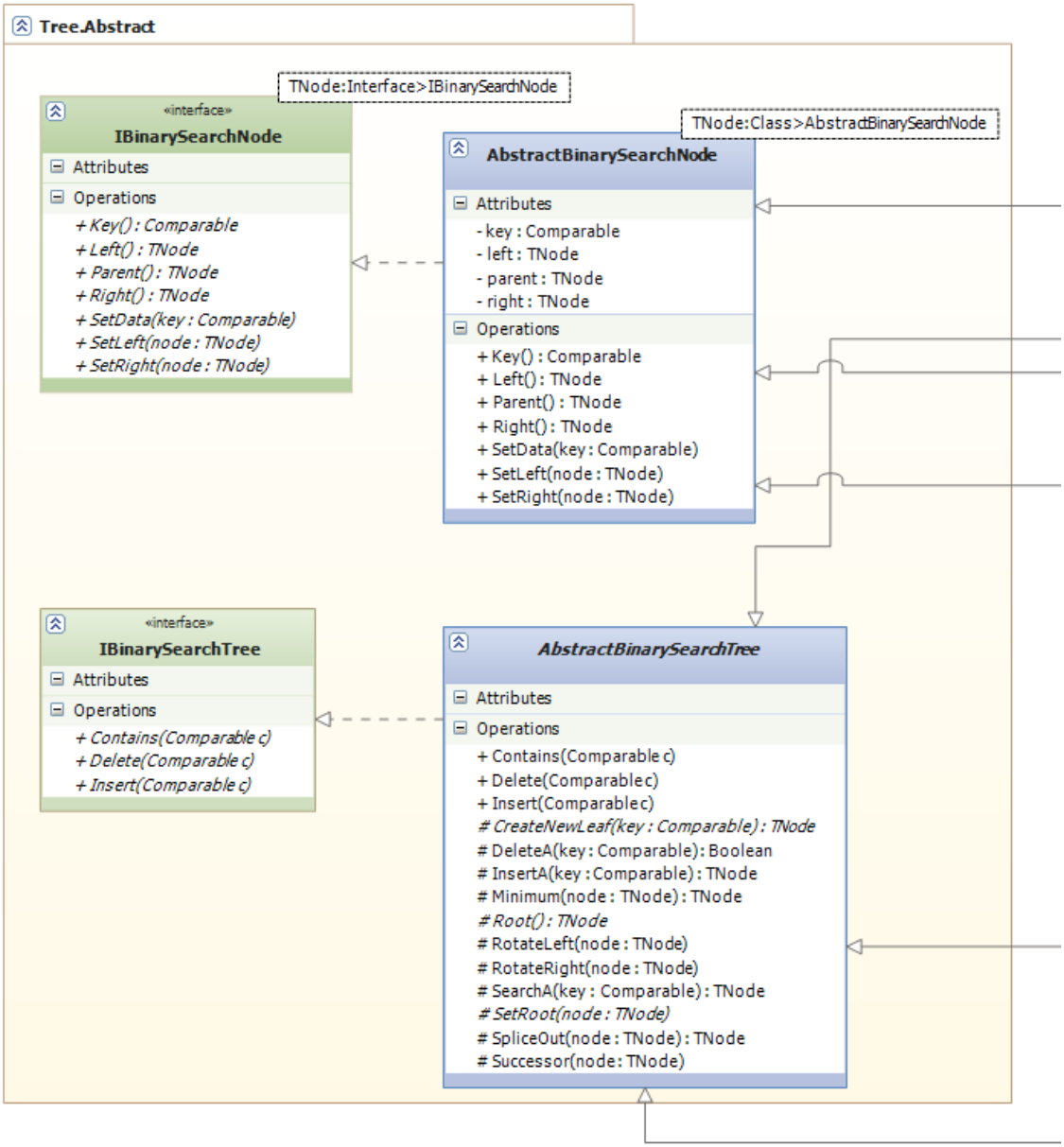


Figura 2.1 Sub-paquete Tree.Abstract del paquete Tree.

Los otros tres sub-paquetes que integran la biblioteca Tree para los Árboles Binarios de Búsqueda se corresponden uno por cada tipo de árbol avanzado de los que se implementan en este trabajo. Entonces existe un sub-paquete Tree.BinarySearch que contiene las clases que implementan al árbol binario de búsqueda; un sub-paquete Tree.Avl que contiene las clases para manejar el árbol binario balanceado AVL y el sub-paquete Tree.RedBlack para el árbol binario Rojo y Negro (RN).

En síntesis no hay mucha diferencia entre estos tres sub-paquetes. Todos contienen dos clases:

1- La clase nodo que hereda de AbstractBinarySearchNode y representa el tipo de nodo correspondiente según el tipo de árbol que sea (BinarySearchNode, AvlNode o RedBlackNode).
2- La clase árbol que hereda de AbstractBinarySearchTree y representa el tipo de árbol binario implementado, es decir, árboles BinarySearchTree, AvlTree o RedBlackTree).

Las tres clases de tipo nodo son muy parecidas, se diferencian en que el nodo RN tiene un atributo booleano llamado ¨color¨ y el nodo AVL tiene un atributo llamado ¨factor balance¨ de tipo entero, los cuales son necesarios en la correspondiente implementación del tipo de árbol. Al mismo tiempo la clase RedBlackNode cuenta con varios métodos relacionados con el color asociado a cada nodo dentro de este tipo de árbol. Por su parte la clase AvlNode tiene varios métodos para manejar el factor de equilibrio en los nodos de este tipo de árbol. En tanto la clase BinarySearchNode solo se compone de los atributos y métodos que hereda de AbstractBinarySearchNode, pues para un árbol binario de búsqueda el nodo no necesita de información adicional.

Por su parte las clases de tipo árbol tienen todos unos atributos raíz el cual es del tipo de árbol al que pertenezcan. Estas clases mantienen la operación Contains que heredan de AbstractBinarySearchTree, pero en los casos de las clases que implementan los árboles AVL y RN se modifican los métodos que realizan las operaciones básicas Delete e Insert, así como se definen algunos métodos protegidos que facilitan el trabajo de dichas operaciones y que deben ser implementados adecuadamente en cada uno de ellos.

En la figura 2.2 puede encontrar un resumen del diagrama de clases que involucran a estos tres sub-paquetes. El diagrama de clases del paquete Tree completo puede consultarse en el Anexo 1.

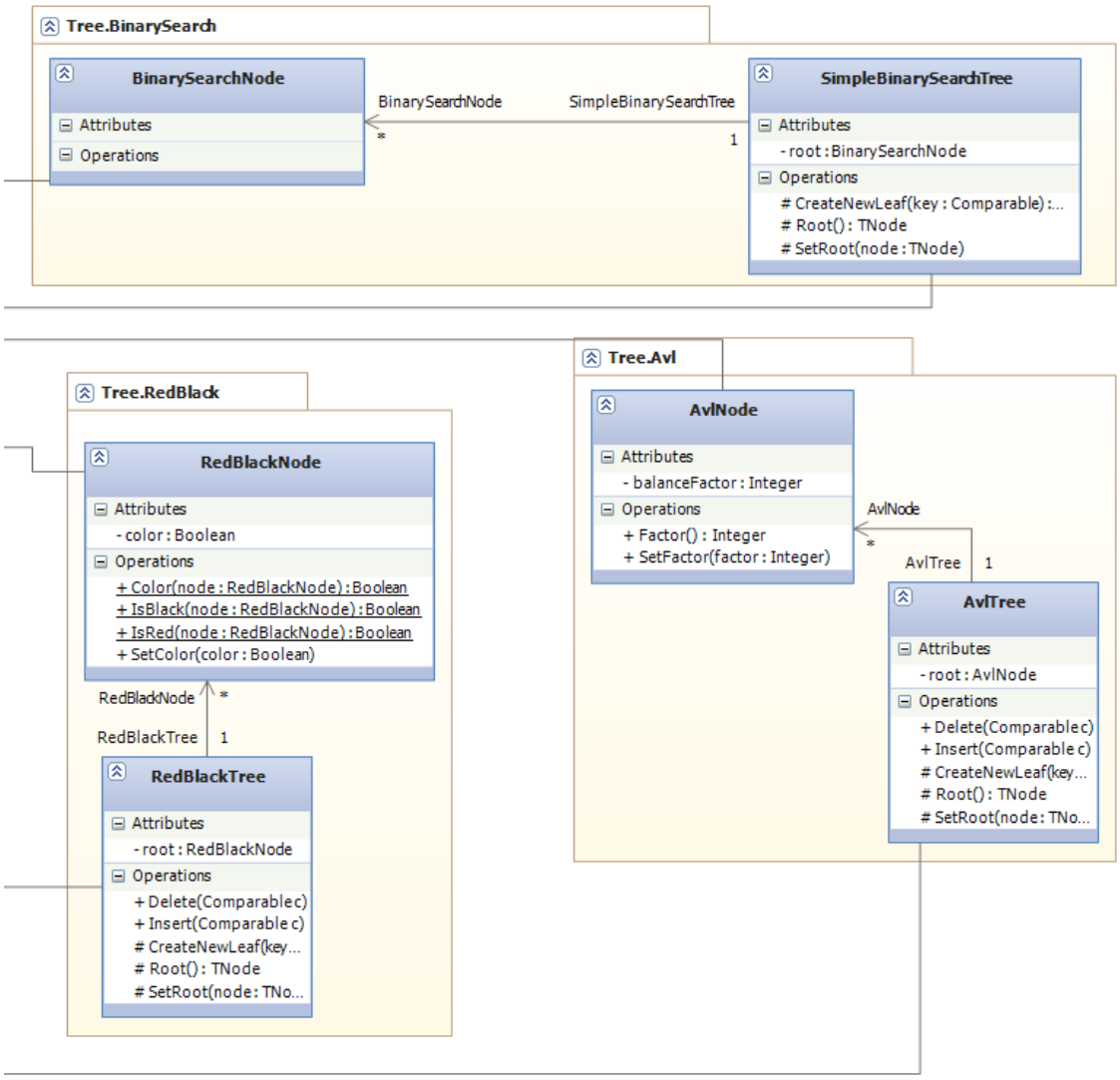


Figura 2.2.Sub-paquetes Tree.BinarySearch, Tree.Avl y Tree.RedBlack del paquete Tree.

2.3 Diagrama de clases del paquete BTree.

En el paquete BTree se implementa un árbol BTree que es persistente en disco gracias a la interfaz IBTreeStorage, y las clases AbstractBTreeStorage y BTreeStorageSample que la implementan. Además, cuenta con una clase BNode que se utiliza para representar los nodos de los árboles BTree. Este nodo BNode utiliza dos veces el TDA Lista a través de la clase ArrayList de Java para manejar una lista con las llaves del nodo y otra con los descendientes del nodo.

En la figura 2.3 se muestran las cinco clases correspondientes al paquete BTree, así como las relaciones entre estas dentro del mismo.

La clase BTree contiene los tres métodos que desarrollan las tres operaciones fundamentales sobre árboles (insertar, buscar y eliminar), y además implementan varios métodos privados para garantizar el funcionamiento de los métodos principales.

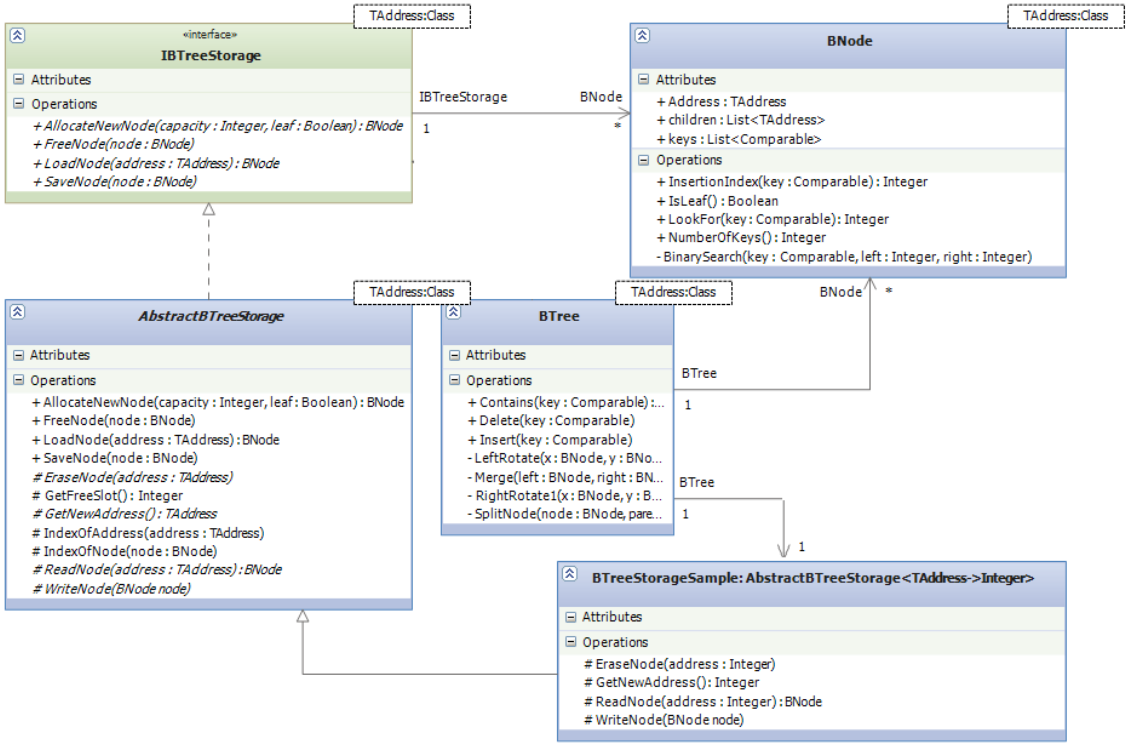


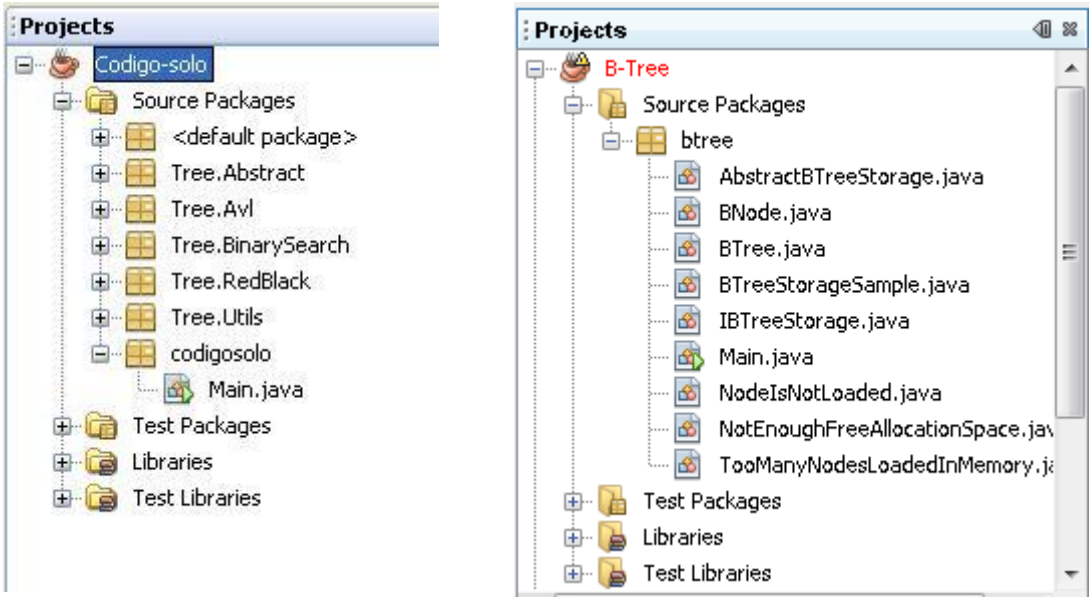
Figura 2.3 Diagrama de clases del paquete BTree.

2.4 Uso y Despliegue del paquete Tree y del paquete BTree.

Una de las intenciones por las que se realiza este proyecto es brindar a estudiantes y profesores el código fuente de una de las formas posibles de implementación de estos tipos de árboles avanzados. Con el código fuente se puede aprender, modificar y agregar elementos a la implementación brindada, cumpliendo con uno de los objetivos a lograr en el aprendizaje de la asignatura de Estructuras de Datos I, según plantea el plan de estudio D. El diseño presentado consta de dos grandes paquetes para que estos sean utilizados en el estudio de los tipos de árboles binarios de búsqueda y los árboles BTree.

Este proyecto fue desarrollado en NetBeans 7.0.1 con Java 7, pero se ha probado que ejecuta correctamente en versiones 6.x de NetBeans para versiones 6.x de Java, por lo que al momento usar el código fuente de estos paquetes en la solución de un problema

dado debemos garantizar la organización del mismo en el conjunto de carpetas que se muestra en la figura 2.4 a) y b) para el paquete Tree y BTree respectivamente.



a) b)

Figura 2.4 Estructura necesaria para el uso: a) del paquete Tree y b) del paquete BTree

Así por ejemplo, en la figura 2.4 a) podemos observar el desarrollo de la aplicación código-solo que usa al paquete Tree para probar con un ejemplo sencillo su uso en la creación de un árbol AVL con las llaves 5, 8 y 15. Ver figura 2.5 donde se muestra el código descrito anteriormente.

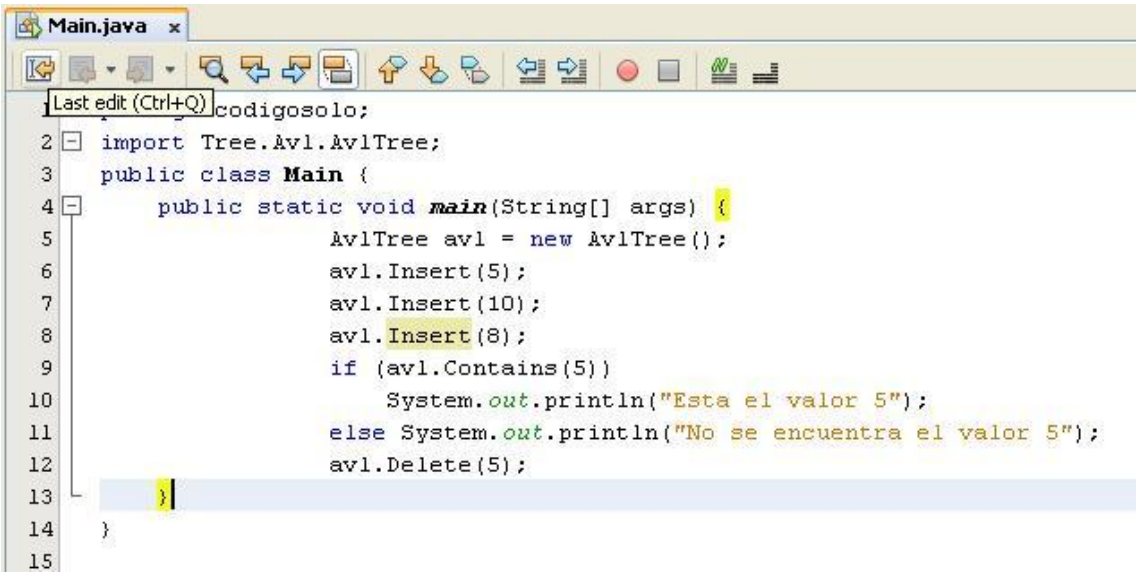


Figura 2.5: Ejemplo de uso del paquete Tree (aplicación código-solo)

Para el despliegue de estos paquetes basta cumplir con los elementos descritos anteriormente para cada una de las computadoras donde se desee utilizar el paquete.

2.5 Diagrama de Actores y Casos de uso de la aplicación *"Demo Animation".*

En este epígrafe se muestra el diagrama actores y casos de uso de la aplicación "Demo Animation" así como una breve descripción de los mismos. Esta aplicación facilita el; aprendizaje de los TDAs árbol binario de búsqueda, árbol AVL y árbol RN, mediante animación de la ejecución de las operaciones fundamentales que se pueden ejecutar sobre esos TDAs.

Con esta aplicación pueden interactuar estudiantes, profesores u otros usuarios interesados en conocer el comportamiento de estos tipos de árboles, contando con posibilidad de realizar todas las acciones del sistema, es decir, seleccionar el tipo árbol, las operaciones a realizar, etc. Para todos estos usuarios están disponibles las mismas funcionalidades. El diagrama de Actores y Casos de Uso de la aplicación se muestra en la figura 2.6 que se muestra seguidamente:

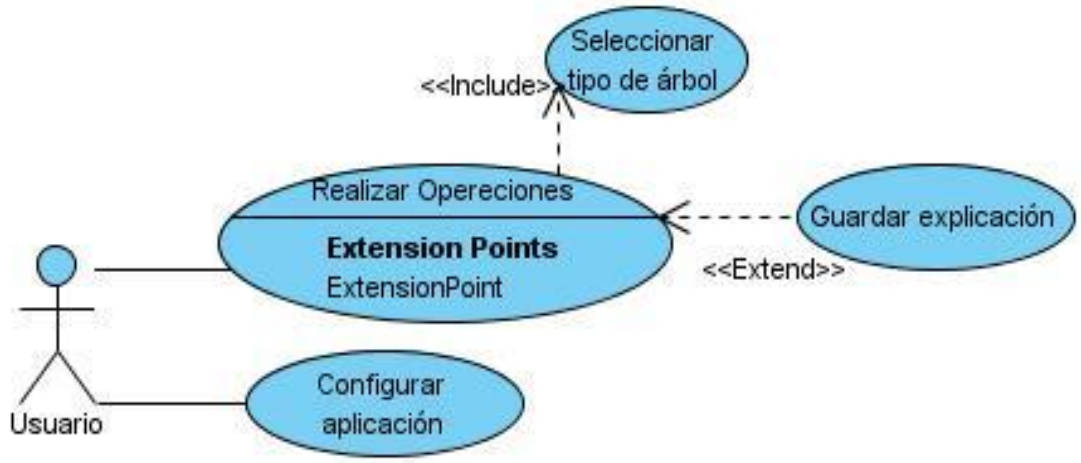


Figura 2.6 Actores y casos de uso de la aplicación "Demo Animation"

2.5.1 Descripción de los casos de uso del sistema.

En este epígrafe se realizará una breve descripción de los casos de uso más significativos de la aplicación.

Caso de uso "Realizar operaciones"

Caso de Uso:	Realizar operaciones
Actor:	Usuario
Propósito:	Realizar todas las operaciones sobre árboles que brinda la aplicación.
Precondición:	
Resumen:	El usuario selecciona una operación y esta se va ejecutando y mostrando en la parte gráfica en dependencia de la configuración del sistema. Se incluye el caso de uso de seleccionar un tipo de árbol.
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
Selecciona un tipo de árbol (activa caso de uso "Seleccionar tipo de árbol")	La indicada en el caso de uso "Seleccionar tipo de árbol"
Escenario 1: El usuario selecciona la operación Insertar dando click en ella.	El sistema genera un número aleatorio, lo muestra e inserta en el árbol en forma gráfica.
Flujo Alternativo de Eventos	
Acción del Actor	Respuesta del Sistema
Escenario 1: Click en la componente JText y edita un número. Click en el botón insertar.	El sistema muestra e inserta en el árbol en forma gráfica el nodo con el valor introducido.
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
Escenario 2: El usuario selecciona la operación Eliminar dando click en ella.	El sistema genera un número aleatorio, lo muestra, lo busca en el árbol. Si no lo encuentra el árbol queda igual, mientras que si lo encuentra lo elimina en el árbol gráfico y realiza los cambios necesarios.

Flujo Alternativo de Eventos	
Acción del Actor	Respuesta del Sistema
Escenario 2: Click en la componente JText y edita un número. Click en el botón eliminar.	El sistema muestra el valor introducido y lo busca en el árbol. Si no lo encuentra el árbol queda igual, mientras que si lo encuentra lo elimina en el árbol gráfico y realiza los cambios necesarios en este.
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
Escenario 3: El usuario selecciona la operación Buscar dando click en ella.	El sistema genera un número aleatorio, lo muestra y lo busca en el árbol. Si lo encuentra el nodo que buscamos se queda encima de su posición en el árbol. Mientras que si no lo encuentra el nodo que buscamos llega hasta la posición en donde debería estar en el árbol.
Flujo Alternativo de Eventos	
Acción del Actor	Respuesta del Sistema
Escenario 3: Click en la componente JText y edita un número. Click en el botón eliminar.	El sistema genera un número aleatorio, lo muestra y lo busca en el árbol. Si lo encuentra el nodo que buscamos se queda encima de su posición en el árbol. Mientras que si no lo encuentra el nodo que buscamos llega hasta la posición en donde debería estar en el árbol.
Acción del Actor	Respuesta del Sistema
Escenario 4: El usuario selecciona la operación Limpiar dando click en ella.	El sistema borra de forma gráfica todo el árbol en el que se está trabajando en el momento que se aplicó la operación.
Flujo Alternativo de Eventos	
Acción del Actor	Respuesta del Sistema

Escenario 4: Click en la componente JText y edita un número. Click en el botón Limpiar.	El sistema borra de forma gráfica todo el árbol en el que se está trabajando en el momento que se aplicó la operación.

Caso de uso "Seleccionar tipo de árbol"

Caso de Uso:	Seleccionar tipo de árbol.
Actor:	Usuario
Propósito:	Escoger un tipo de Árbol.
Precondición:	
Resumen:	El usuario selecciona un tipo de árbol que será escenario de las operaciones que le realizará el caso de uso ***"realizar operaciones"*** que lo incluye.
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
Click en el botón correspondiente al tipo de árbol que desea seleccionar.	Borra el gráfico del árbol actual y muestra el gráfico del nuevo tipo de árbol seleccionado.

Caso de uso "Configurar aplicación."

Caso de Uso:	Configurar aplicación.
Actor:	Usuario
Propósito:	Usar alguna de las opciones que brinda el sistema.
Precondición:	
Resumen:	El usuario selecciona una opción y esta la aplica sobre el árbol que se esté usando en ese momento.
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
Escenario 1: Click en el	Realiza las siguientes operaciones del sistema, sobre el

botón pausar para activarlo.	árbol en cuestión, paso a paso.
Escenario 2: Click en el botón pausar para desactivarlo.	Realiza las siguientes operaciones del sistema, sobre el árbol en cuestión, de una sola vez.
Escenario 3: Click en el botón Autoescalar para activarlo.	Ajusta el árbol de la parte gráfica de forma que siempre en su totalidad por mucho que este crezca.
Escenario 4: Click en el botón Autoescalar para desactivarlo.	Si el árbol de la parte gráfica estaba ajustado por su gran tamaño, se desajusta y solo muestra lo que quepa en esa área y activando la barra de desplazamiento para moverme hacia los lados del árbol.

Caso de uso "Guardar explicación"

Caso de Uso:	Guardar explicación
Actor:	Usuario
Propósito:	Guarda la explicación de la última operación que se realizó.
Precondición:	
Resumen:	Se salva un txt de la explicación de la operación que se realizó.
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
Click en el botón Guardar	Aparece el dialogo de guardar para escoger la posición en el disco y el nombre donde se desee guardar la explicación.

2.6 Diagrama de clases del paquete *"TreeAnimationApp1"* para la parte visual de la aplicación *"Demo Animation"*.

En este paquete se han programado todas las clases que tienen que ver con la visualización de las operaciones de los árboles y estas a su vez interactúan con clases de los paquetes descritos anteriormente y con la clase TreeAnimationApp1App que es la

clase principal (contiene la función main) y que solo activa a una instancia de la clase TreeAnimationApp que es la ventana principal de la aplicación.

La clase TreeAnimationApp contiene como atributos instancias de cada uno de los tipos de árboles a visualizar, esto nos permite mantener una representación de cada tipo árbol a visualizar y poder intercambiar visualmente de uno a otro por las opciones de la ventana principal sin perdida de información. También tiene una instancia de la clase AminationManager que es la responsable de realizar la visualización y animación de las operaciones sobre cada uno de los tipos de árboles.

La clase AnimationManager almacena toda la información de los nodos a visualizar de cada uno de los árboles activos, usando una estructura de datos llamada HashMap implementada entre las colecciones de Java. La información de los nodos a visualizar esta formada por instancias de la clase DrawableNode y NodeLayout.

La clase DrawableNode contiene la información de los nodos a visualizar: coordenadas (x, y), color del fondo, de la letra, del borde, radio del circulo y una referencia a los nodos hijos.

La clase NodeLayout contiene información para la identificación del nodo y del diseño visual del mismo durante la animación de las operaciones. Esta clase utiliza la clase Cell para representar una celda rectangular asociada a cada nodo visualizado. Una instancia de la clase Frame, maneja todos los NodeLayout necesarios para mostrar la información visual de los árboles de manera correcta, a través de un HasMap.

La clase TreeAnimationAppView maneja una instancia de la clase Frame para tener acceso a las caracteristicas de los nodos a dibujar y tambien maneja un JPanel llamado TreeArea que es donde se dibujan los 3 tipos de árboles, uno a la vez. Cada nodo a dibujar es un objeto de la clase DrawNode.

A continuación en la figura 2.7 se muestra el diagrama de clases que muestra las asociaciones entre las clases del paquete TreeAnimationApp1.

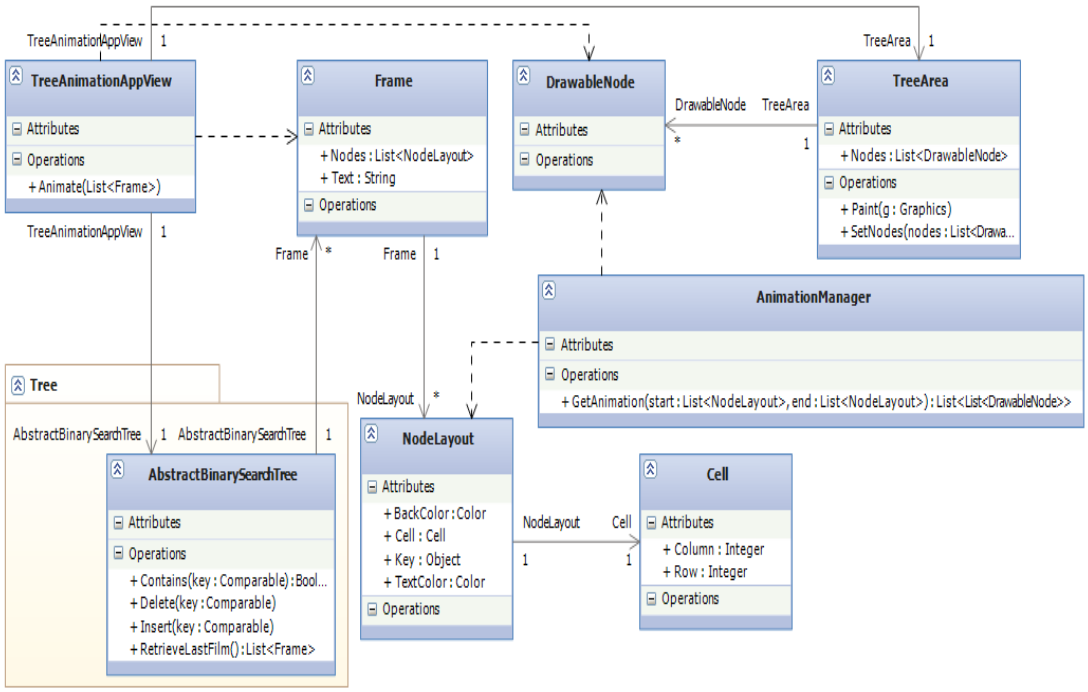


Figura 2.7 Diagrama de clases del paquete ***"TreeAnimationApp"*** para la parte visual de la aplicación **"Demo Animation"**

2.7 Uso y despliegue de la aplicación visual *"Demo Animation".*

En este epígrafe se da una breve descripción del uso y despliegue de la aplicación "Demo Animation". Los diagramas de despliegue muestran las relaciones físicas entre los distintos nodos que componen un sistema y la distribución de los componentes sobre dichos nodos.

La aplicación para el trabajo con árboles "Demo Animation" esta implementada en NetBeans 7.0.1 y utiliza una versión del paquete Tree que apoya la representación visual del comportamiento de las operaciones sobre los árboles estudiados. Esta aplicación se distribuye a través del archivo TreeAnimationApp1.jar el cual debe ser copiado en alguna carpeta de trabajo del usuario que desee explotar las funcionalidades de esta aplicación. Para que la aplicación contenida en TreeAnimationApp1.jar ejecute correctamente debe estar instalado la máquina virtual de Java (JRE) en su versión 6 o versiones superiores y el usuario debe activarlo presionado doble click sobre el archivo.

Capítulo 3: Manual de usuarios del software *Trabajo con Árboles- Demo Animation.*

El presente capítulo está dedicado a los requerimientos de hardware y de software necesarios para lograr un rendimiento óptimo del Sistema, además de un manual de usuario donde se explican los pasos a seguir por parte del mismo para la correcta utilización de todas las funciones del software. Así como una guía de cómo debe usarse dicha aplicación para garantizar un mayor aprendizaje en la asignatura Estructuras de Datos y Algoritmos I y II.

3.1 Requerimientos del software Trabajo con Árboles- Demo Animation

Para el correcto funcionamiento del software se necesitan un mínimo de requerimientos técnicos tanto de hardware como de software.

Requerimientos de hardware:

- Al menos 64 MB de memoria RAM.
- Se necesita de 150 MB para la instalación del RunTime Enviroment (JRE) Versión 6 de Java.
- Computador Pentium de 266MHz o superior.

Requerimientos de software:

- Sistema Operativo Windows 2000, XP, 2003, Vista (x86, x64), Windows 7 (x86, x64) .
- Sistema Operativo Linux que tenga instalado alguno de los siguientes administradores gráficos de ventanas para X: Common Desktop Environment (CDE), GNOME, The K Desktop Environment, Xfce Desktop Environment.
- Máquina virtual de Java (JRE) en su versión 6 o versiones superiores.

3.2 Utilización del software Trabajo con Árboles- Demo Animation

A continuación ofrecemos una guía de cómo utilizar el software ***Trabajo con Árboles- Demo Animation***.

El primer paso es ejecutar ***TreeAnimationApp1.jar*** (ver Figura 3.1 con la ventana principal del software).

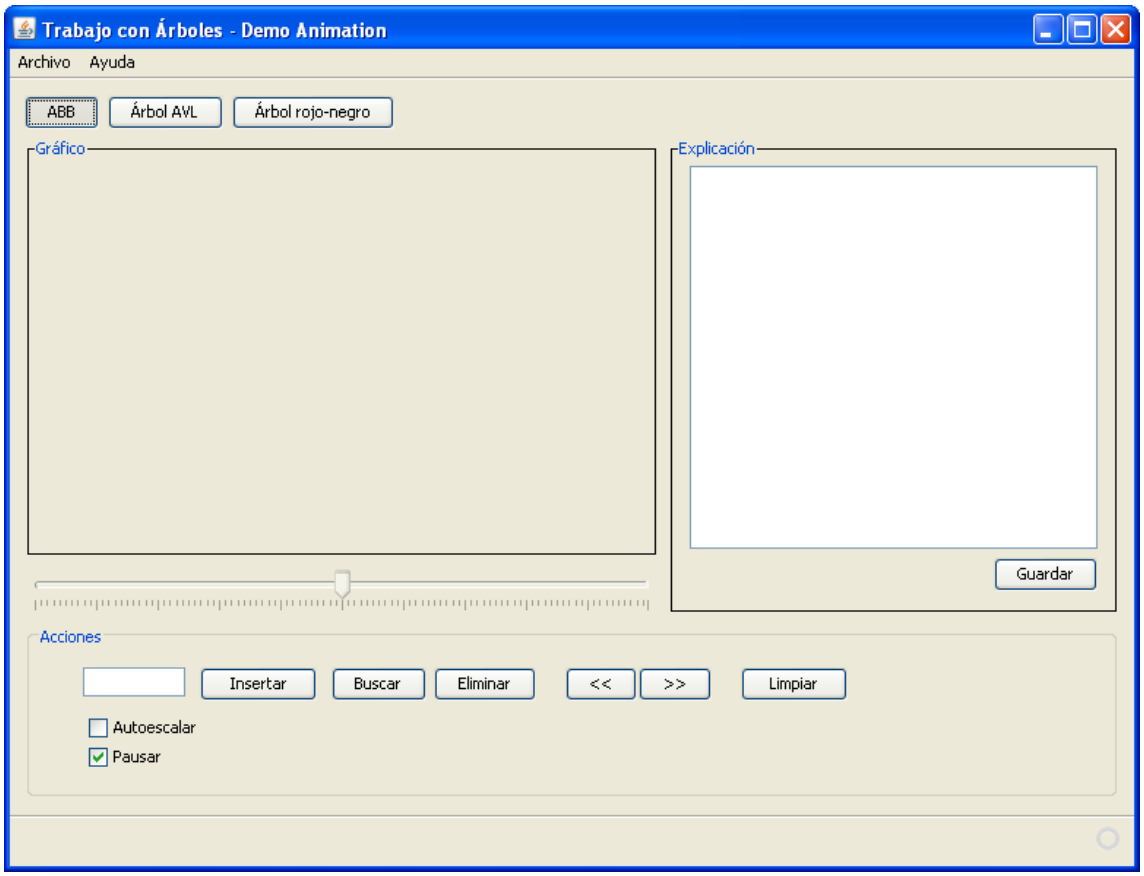


Figura 3.1 Ventana principal del software.

El software da la posibilidad de representar las principales operaciones ***Insertar***, ***Buscar***, ***Eliminar*** en 3 tipos de árboles binarios de búsqueda: ABB, Árboles AVL y Árboles rojo-negro (Árboles r-n).

En la Figura 3.2 se muestra un esquema de las diferentes componentes por las que está formada la ventana principal del software. El menú de la aplicación se muestra en el área sombreada en amarillo. La parte señala en rojo muestra los 3 tipos de árboles binarios de búsqueda que se pueden representar y sus respectivos gráficos deben aparecer en la parte señalada con azul. En la componente señalada en verde se brinda una explicación de todos los pasos que se realizan cuando se efectúa una operación de ***Insertar***, ***Buscar*** o ***Eliminar***; además la misma cuenta con un botón ***Guardar*** que da la posibilidad de salvar en un documento texto dicha explicación para estudios posteriores. En la parte inferior señalada en rosado se muestran las distintas operaciones que se pueden realizar con los distintos tipos de árboles ya sea de forma aleatoria(o sea dejando el espacio en blanco) o insertando un valor. Esta última componente brinda también la opción de ***Pausar*** (que permite observar la animación de una operación paso a paso, ya

sea hacia adelante o hacia atrás mediante los botones << >>), ***Autoescalar*** (que ajusta el tamaño del árbol en el área de dibujo según este vaya creciendo) y ***Limpiar*** (que se ocupa de borrar automáticamente el área de dibujo).

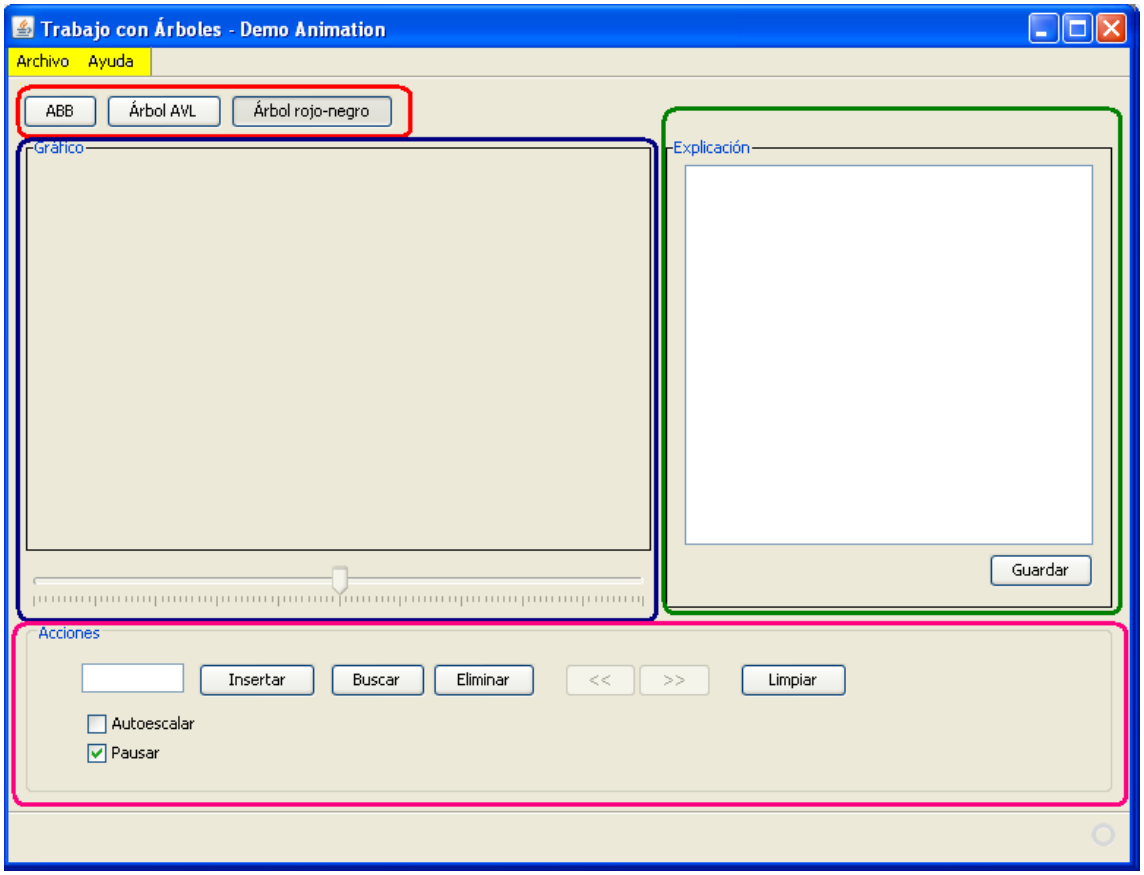


Figura 3.2 Ventana principal del software dividida en componentes.

En la Figura 3.3 se muestra un ejemplo de un árbol ABB con un solo nodo creado de forma aleatoria y la explicación de dicha operación lista para guardarse en documento texto.

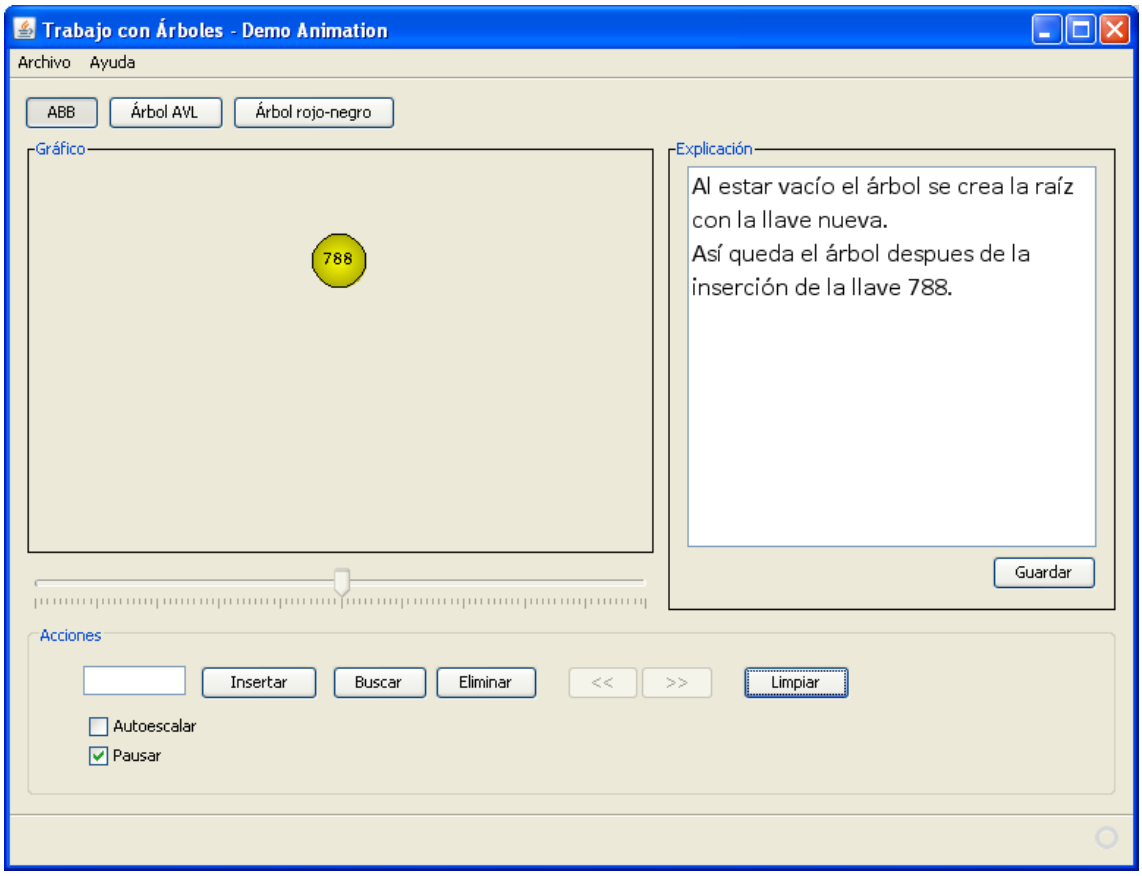


Figura 3.3 Representación de un árbol ABB con solo un nodo.

En el caso de los árboles AVL se utilizó el criterio de asignar un color a cada nodo para facilitar al usuario la representación visual del factor equilibrio del mismo, esto es:

rojo: factor equilibrio +2 (desbalance hacia la derecha).

azul fuerte: factor equilibrio -2 (desbalance hacia la izquierda).

amarillo: factor equilibrio +1 (semi-desbalance hacia la derecha).

azul claro: factor equilibrio -1 (semi-desbalance hacia la izquierda).

verde: factor equilibrio 0 (perfectamente balanceado).

La Figura 3.4 y 3.5 muestran un árbol AVL con 5 nodos, donde 3 fueron creados de forma aleatoria y 2 insertando los valores 56 y 45. En el caso de la Figura 3.4 el gráfico muestra un árbol desbalanceado hacia la izquierda tras la inserción del nodo 45 y la Figura 3.5 muestra cómo queda el árbol finalmente luego de las rotaciones necesarias para la recuperación del balance del árbol.

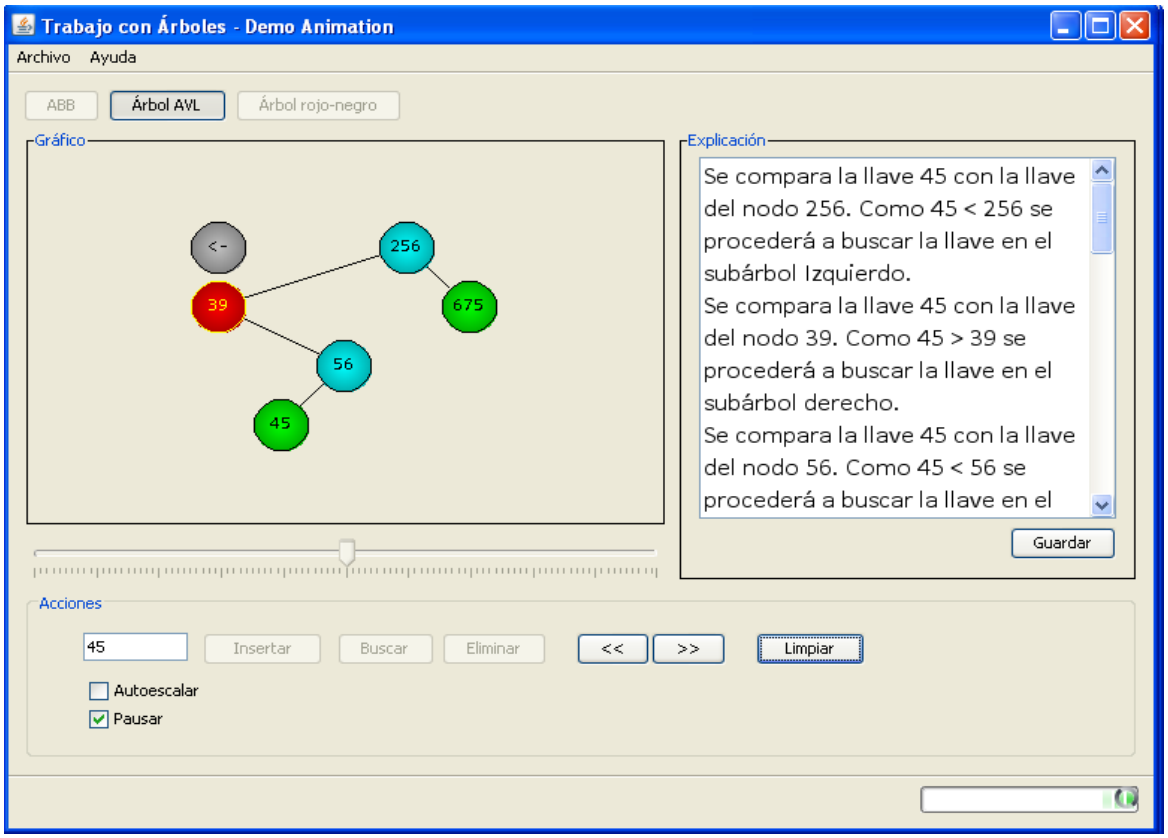


Figura 3.4 Representación de un árbol AVL antes de balancearse luego de un inserción.

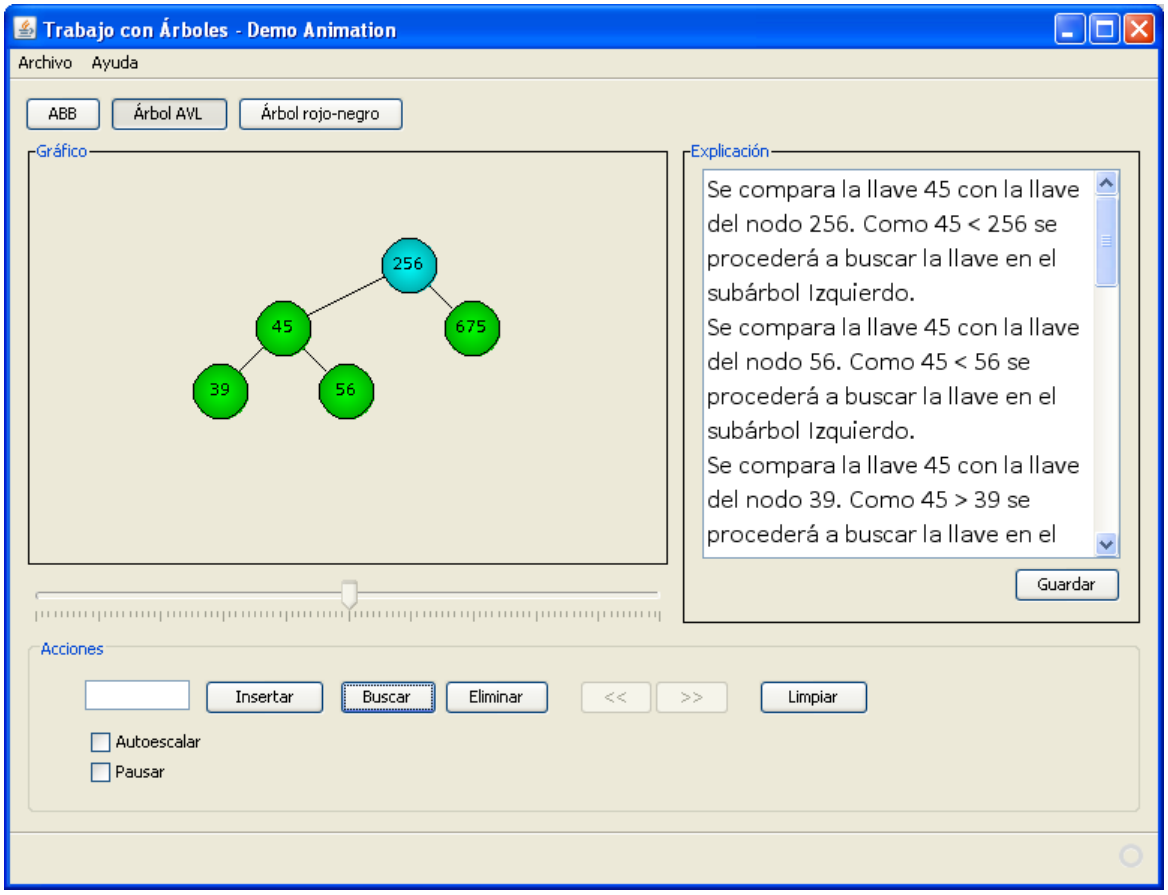


Figura 3.5 Representación de un árbol AVL balanceado luego de una inserción.

En la Figura 3.6 se muestra un árbol R-N (Rojo-Negro) con 9 nodos generados en el siguiente orden 23, 35, 42, 38, 29, 15, 80, 40, 41 y luego eliminar 41 y 23 e insertar un nodo de forma aleatoria.

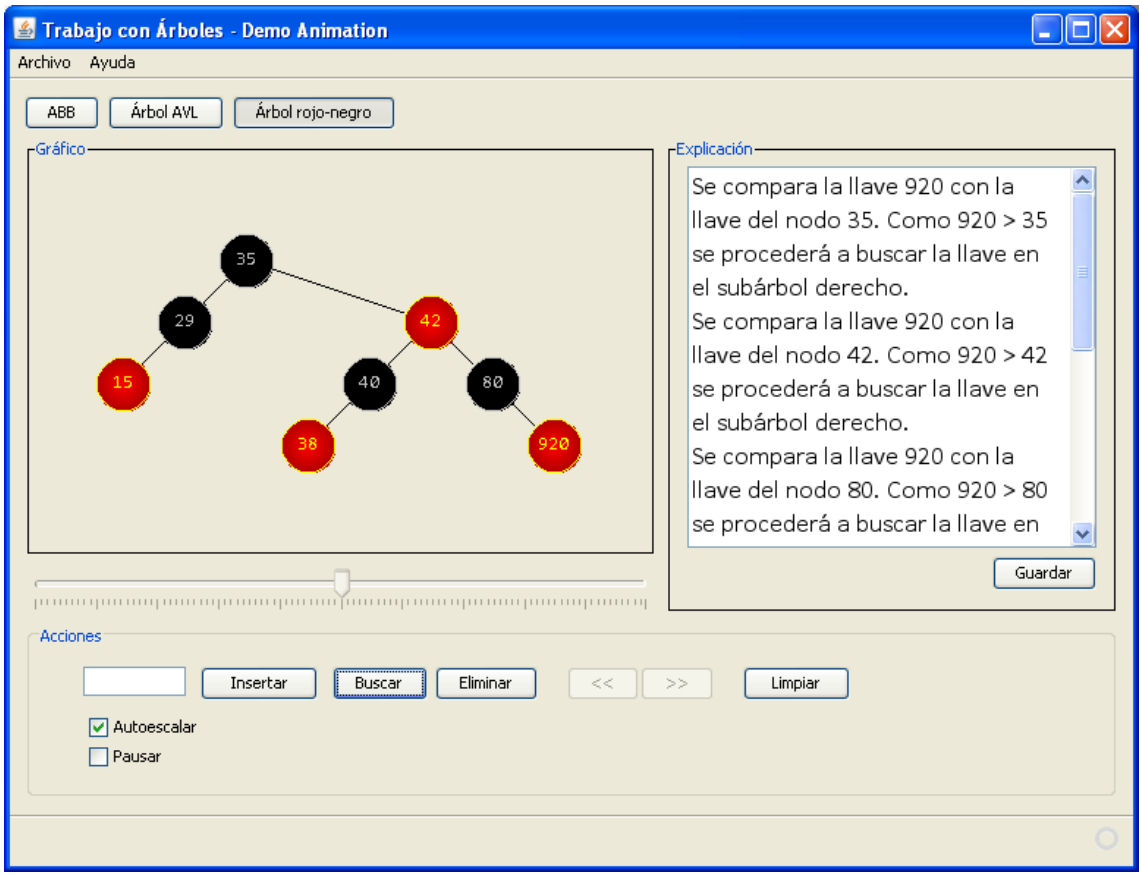


Figura 3.6 Representación de un árbol R-N luego de una secuencia determinada de inserciones y eliminaciones.

Una opción interesante que presenta la aplicación es que deja guardada en memoria el trabajo que hayas realizado con algún tipo de árbol específico y tras moverte hacia otro de estos y regresar al que estabas trabajando en un inicio, puedes contar aún con el gráfico que tenías de dicha estructura de datos.

Por último pero no menos importante, la aplicación brinda la posibilidad de que al situar el mouse encima de un nodo este se enmarque en un cuadro de rayas discontinuas y al aplicar doble click sobre este cumple la función de eliminar el nodo y todos los cambios que esto provoca. La Figura 3.7 muestra un árbol BB en el cual el nodo 327 se encuentra en dicha situación tras la que puede ser eliminado por el usuario.

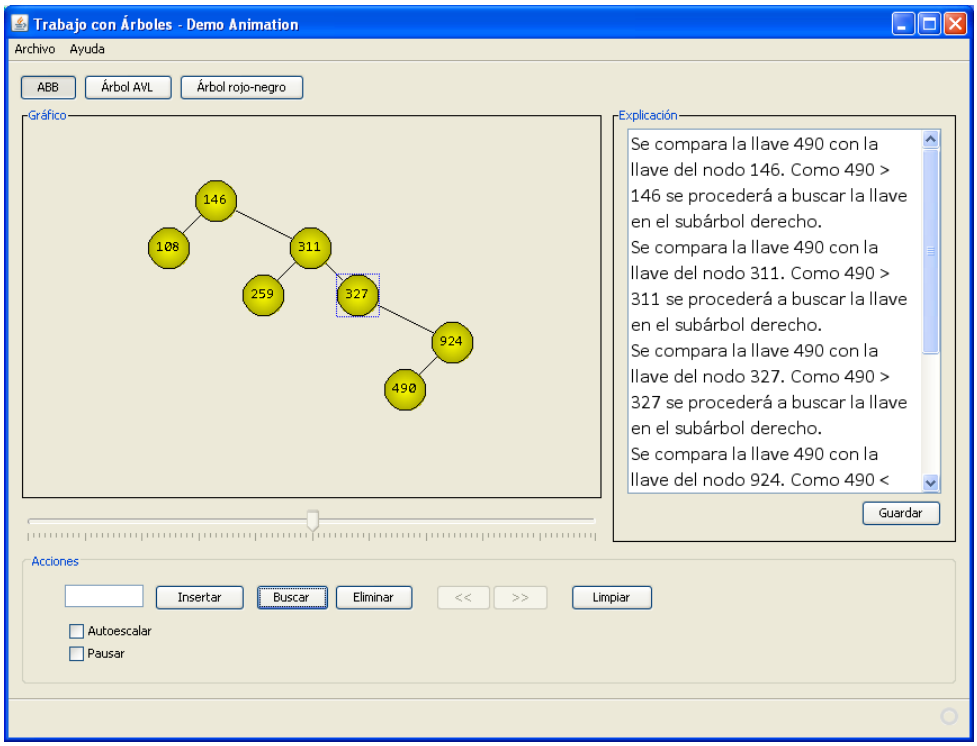


Figura 3.7 *Árbol BB* con un nodo enmarcado por la presencia del mouse.

3.3 Uso de las herramientas desarrolladas en la docencia.

En este epígrafe se pretende brindar a estudiantes, profesores y demás interesados una forma de utilizar todas las herramientas que presenta este proyecto investigativo para garantizar un mayor aprovechamiento de estos; y sobretodo que facilite y guie el estudio de los tipos de datos abstractos (TDA) árboles binarios de búsqueda (ABB) y sus versiones más avanzadas como los AVL y los ARN para esta primera versión de la aplicación, en la asignatura Estructuras de datos I en las carreras de Ciencia de la Computación e Ingeniería Informática.

3.3.1 Uso de los códigos fuentes de los paquetes Tree y BTree en la asignatura.

Para el uso de este código fuente en la asignatura, deben ofrecerse los directorios que contienen los paquetes *Tree* y *BTree* como se explicó en el epígrafe 2.4, para que este sea utilizado por los estudiantes en el transcurso de la asignatura como la implementación básica de los TDA árboles binarios de búsqueda.

Luego de que los estudiantes cuenten con los códigos fuentes originales los profesores deberán basar sus conferencias y clases prácticas con los mismos criterios de implementación que se usaron en el desarrollo de la herramienta para mantener la

coherencia a la hora de referirse al mismo. Basándose en las técnicas de programación utilizadas y las clases y métodos ya desarrollados, el profesor debe enfocar a los estudiantes en aprender y entender cómo se programaron dichas clases. Además atendiendo a que por supuesto esta no es la única forma de representación de estos árboles se debe orientar por parte del profesor la modificación del código fuente para ver diferentes formas de implementar las operaciones, siempre haciendo copia de las originales para no perderlas.

También se generó un javadoc de la aplicación con comentarios en Java para describir las clases y métodos de la misma, así como los parámetros y valores que devuelven.

3.3.2 Uso de la aplicación "Demo Animation" en la asignatura.

Para el uso de este demo el profesor debe facilitarles a los estudiantes el .jar de la aplicación para su uso en los laboratorios de la asignatura. Luego debe orientar para su uso diferentes juegos de datos donde ocurran situaciones interesantes para que el estudiante observe de qué manera se soluciona en la animación, ya sea de forma rápida o con la opción de *pausar* activada para ir viendo los pasos de la operación.

Por ejemplo un estudio orientado por el profesor puede ser realizar en la aplicación *"Demo Animation"* la siguiente secuencia de la operación *insertar* para un *árbol RN* (146,372,855,123,677,150,149,148,147) y observar con la opción *pausar* que sucede cuando se inserta el último nodo de esta secuencia. La Figura 3.9 muestra el árbol en el momento en que se inserta el nodo 147 en la posición correspondiente, en este se puede notar que incumple una de las propiedades de los *árboles RN* en la que dice que si un nodo es rojo sus hijos deben ser negros.

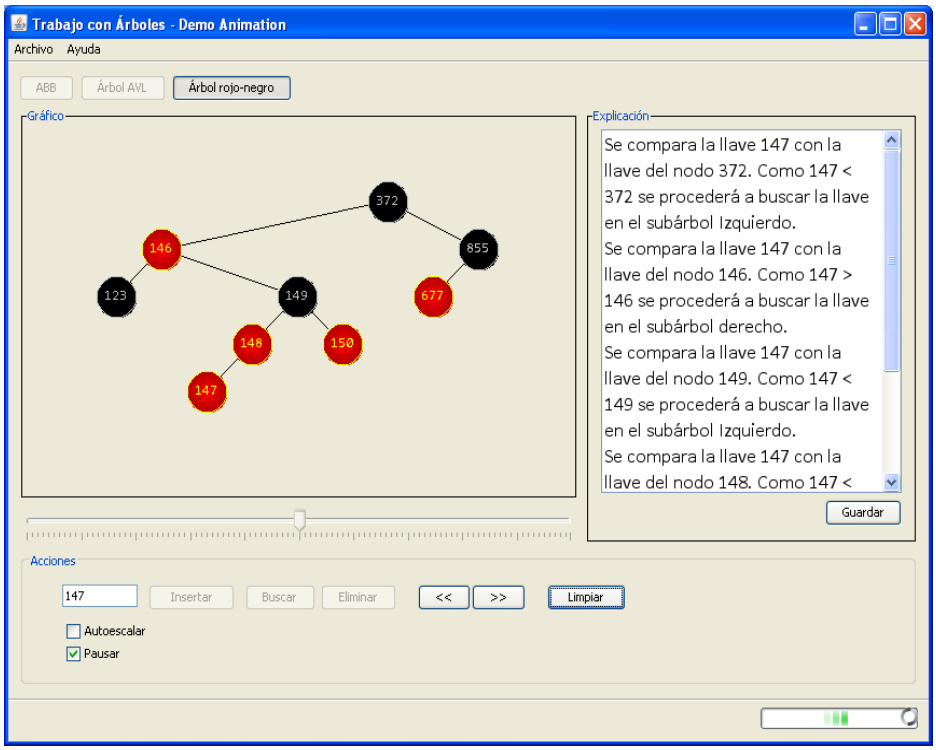


Figura 3.9 ***Árbol RN*** en el momento que se ***inserta*** el nodo 147.

Para resolver esta situación se procede a hacer un cambio de color de forma ascendente desde la hoja hacia arriba hasta solucionar el problema. En nuestro ejemplo específico el estudiante puede observar en el demo que luego del cambio de color se vuelve a incumplir esa misma propiedad en la que además de un cambio de color es necesario realizar una rotación doble (izquierda-derecha) entre los nodos 372-146-149 con la cual se restablecen todas las propiedades del árbol. La Figura 3.10 muestra cómo queda el árbol luego de los cambios necesarios en esta operación.

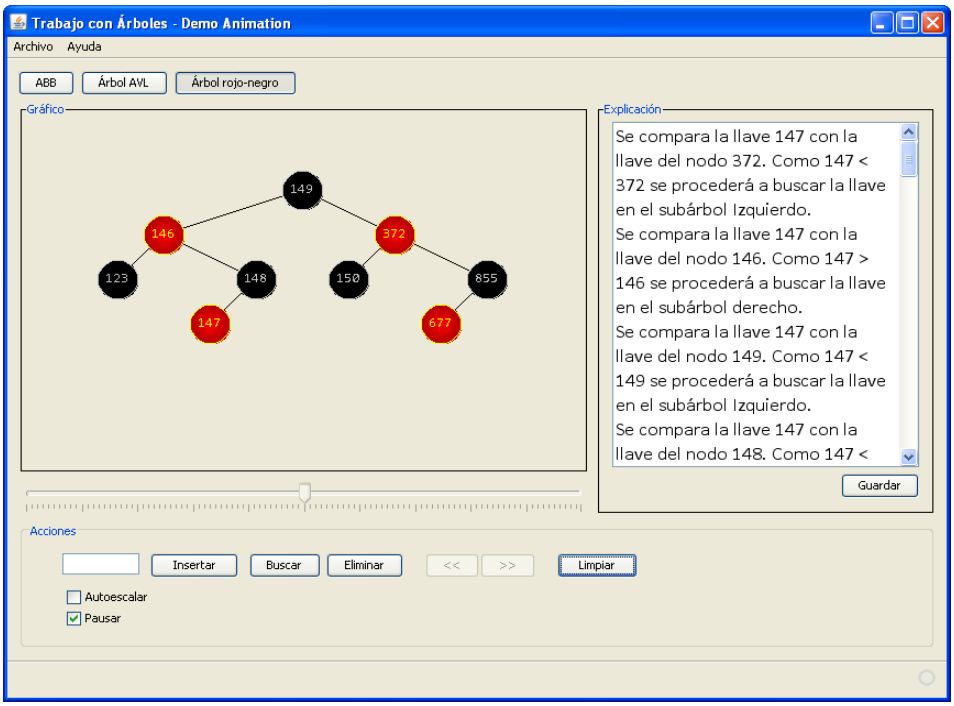


Figura 3.10 ***Árbol RN*** luego de que se restablecieron sus propiedades tras la operación de ***insertar*** el nodo 147.

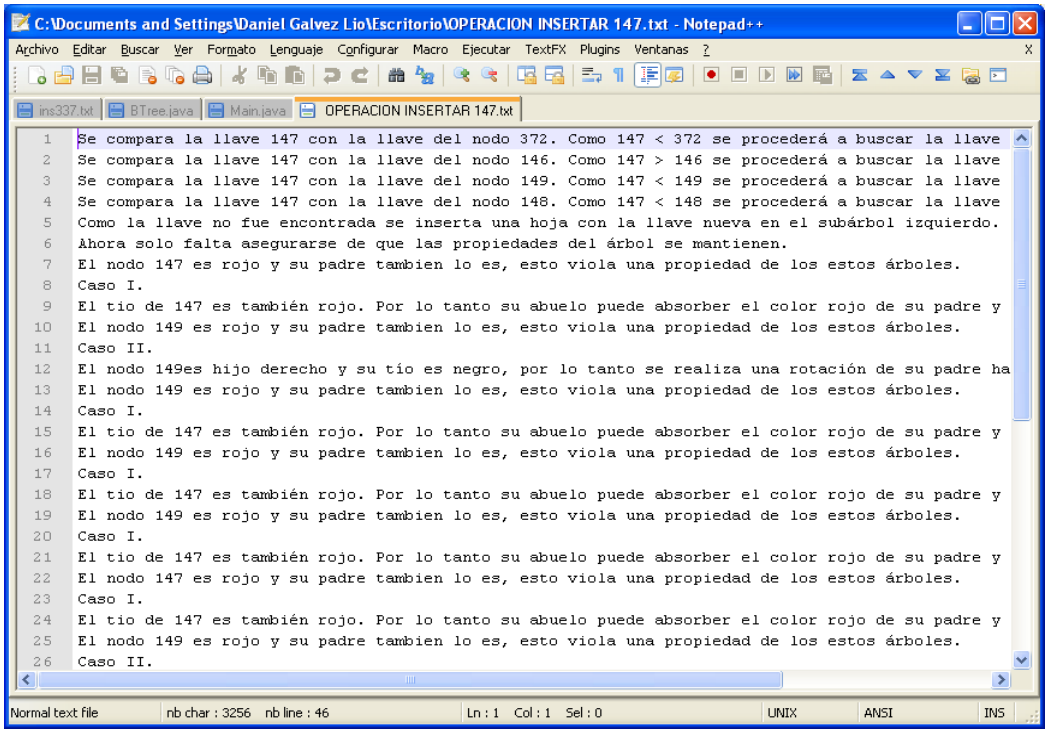


Figura 3.11 Documento texto de una operación insertar generada por la operación ***Guardar***

Además de ir observando paso a paso los cambios en el árbol en una operación, el estudiante puede utilizar la opción ***Guardar*** en la explicación para salvarla en un documento de texto y obtener algo así como lo que se muestra en la Figura 3.11; esto

puede copiarlo en cualquier dispositivo de almacenamiento como una memoria flash y llevárselo para estudiarlo mejor en alguna PC donde ni siquiera necesita tener instalado la Máquina virtual de Java (JRE).

Conclusiones

El plan de estudio D de la carrera de Ciencia de la Computación recoge en la asignatura de Estructura de Datos y Algoritmos I un grupo de objetivos instructivos y habilidades a desarrollar por los estudiantes para el trabajo con los TDA árbol binario de búsqueda, árbol AVL y árbol rojos y negro que propician la necesidad de dominar y poseer el código fuente de alguna implementación de esos TDA, así como dominar la implementación de las operaciones básicas: insertar, eliminar y buscar.

Se propone una solución formada por a) los paquetes con los códigos fuente que también se distribuyen en .jar, lo cual permite a los estudiantes, bajo orientación del profesor, estudiar y modificar los códigos de las operaciones básicas de los TDA en estudio; b) una aplicación visual que representa visualmente las operaciones básicas de los TDA en estudio.

Se realizó una implementación en Java de la solución propuesta, obteniéndose los paquetes Tree y BTree y la aplicación visual de apoyo a la docencia para mostrar cómo se realizan las operaciones básicas para estos TDA en estudio.

Se realiza una propuesta de uso de estas herramientas como apoyo a la docencia de la asignatura EDA I.

Recomendaciones

- Extender la aplicación visual para el TDA árbol BTree.
- Desarrollar un estudio y la implementación correspondiente de los TDA QuadTree y OcTree.
- Valorar la posible utilización de estos resultados en la carrera de Ingeniería Informática

Bibliografía

Allen, M. (2004) *Estructuras de Datos en Java™ Compatible con Java™ 2*. Editorial Félix Varela, La Habana.

Anderson, A. (1993) Balanced Search Trees Made Simple. [en línea] disponible en: http://www.ganimides.ucm.cl/haraya/doc/10.1.1.118.6192.pdf, accedido el 13 de agosto de 2012.

Araya, C. (2012) "Árboles B" [en línea] disponible en http://www.ganimides.ucm.cl/haraya/doc/Árboles_B_2.pdf, accedido el 17 de julio de 2012.

Bayer, R. (1972) "Symmetric binary B-trees: Data structure and maintenance algorithms." Acta Informatica 1:290-306.

Canales Villacrés P. A. (2008) "Árboles", [en línea] disponible en http://peruforo.com/index.php/topic,88.0/prev_next,prev.html#new, accedido el 17 de julio de 2012.

Pozo Coronado, S. (2002) "Capítulo 8 Árboles AVL". [en línea], disponible en: http://c.conclase.net/edd/?cap=008#inicio, accedido el 16 de julio de 2012.

CECS (2012) "B^+-Tree Indexes: Storage structure of records", [en línea], disponible en: http://www.cecs.csulb.edu/~monge/classes/share/B+TreeIndexes.html, accedido el 30 de Julio de 2012.

Culberson, J. y Munro, J. (1990) "Analysis of the Standard Deletion Algorithms in Exact Fit Domain Binary Search Tree", [en línea] disponible en: http://www3.ime.usp.br/~yoshi/2012i/mac323/exx/ST_ABB_Evolution/culberson_Algo.pdf, accedido el 13 de agosto de 2012.

Culberson, J. y Munro, J. (1990) "Explaining the behavior of binary search trees under prolonger updates: a model and simulations." [en línea], disponible en: http://comjnl.oxfordjournals.org/content/32/1/68.full.pdf, accedido el 13 de agosto de 2012.

De la Torre, A "Estructuras de Datos y Algoritmos en Java". [en línea], disponible en: http://www.programacion.com/articulo/estructuras_de_datos_y_algoritmos_en_java_309/6, accedido el 16 de julio de 2012.

EDD (2000) “Capítulo 7 Árboles binarios de búsqueda (ABB)”, [en línea] disponible en: http://c.conclase.net/edd/?cap=007, accedido el 16 de julio de 2012.

Fernández, J. (2012) “Métodos de inserción y de búsqueda”. [en línea] disponible en: http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/docs/arb_B.htm, accedido el 17 de julio de 2012

Gálvez Rojas, S. (2012) “Árboles AVL y heaps” Universidad de Málaga. [en línea] disponible en: http://www.lcc.uma.es/~galvez/ftp/tad/tadtema4cont.pdf, accedido el 17 de julio de 2012.

Gómez, I. (2012) “B-árboles”, [en línea] disponible en: http://ict.udlap.mx/people/ingrid/Clases/IS211/Árboles.html, accedido el 17 de julio de 2012.

González, A. J. (2012) “Árboles de Búsqueda Binaria”. [en línea] disponible en http://profesores.elo.utfsm.cl/~agv/elo320/01and02/dataStructures/binarySearchTree.pdf, accedido el 17 de julio de 2012.

González Harbour, M. (2012) “Estructuras de datos y algoritmo” Universidad de Cantabra. [en línea] disponible en: http://www.ctr.unican.es/asignaturas/eda/cap3-jerarquicos-2en1.pdf, accedido el 17 de julio de 2012.

Gutiérrez Carreón, G. (2010) “Árboles” ___, Universidad Nacional del Nordeste. Argentina, [en línea], disponible en:http://www.fcca.umich.mx/descargas/apuntes/Academia%20de%20Informatica/Estructura%20de%20Datos%20%20%20%20G.a.G.C/Unidad%206.pdf , accedido el 17 de julio de 2012.

ICT, (2012) “Estructuras de datos dinámicas/Árboles”, [en línea], disponible en: http://ict.udlap.mx/people/ingrid/Clases/IS211/Árboles.html, accedido el 16 de julio de 2012.

Lynch, W. (2012) “More combinatorial properties of certain trees". [en línea] disponible en: http://mjnl.oxfordjournals.org/content/7/4/299.full.pdf, accedido el 13 de agosto de 2012.

Marín, P. (2012) “Representación de conjuntos mediante árboles.” disponible en: http://dis.um.es/~nmarin/sec4.4.pdf, accedido el 17 de julio de 2012.

Marín Pérez, N. (2012) "Árboles" Universidad de Murcia. [en línea] disponible en http://usuarios.multimania.es/árbolesbpro/queesunarbolb.htm

Martínez López, P. (2012) "Estructuras de datos" URN. [en línea] disponible en: http://www.fceia.unr.edu.ar/lcc/t312/archivos/ED-02.árboles.pdf, accedido el 17 de julio de 2012.

Meneses E. (2012) "Árboles Rojo-Negro". [en línea] disponible en http://www.oocities.org/emenesesr/recursos/árbolesRojoNegro.pdf, accedido el 17 de julio de 2012.

MES, (2007) "Programa de la asignatura Programación e Ingeniería de Software". Universidad de La Habana.

Morris, J. (1998) "Estructuras de Datos y Algoritmos: Árboles AVL", [en línea] disponible en: http://translate.google.com.cu/translate?hl=es&langpair=en|es&u=http://www.cs.auckland.ac.nz/~jmor159/PLDS210/AVL.html, accedido el 16 de julio de 2012.

OCW (2012) "Estructuras de Datos", [en línea] disponible en http://ocw.upm.es/lenguajes-y-sistemas-informaticos/estructuras-de-datos/contenidos/tema4nuevo/Árboles.pdf, [en línea] accedido el 16 de julio de 2012.

Peral CortéS, J. (2009) "Tema 3 El tipo árbol. Programación y estructuras de datos". [en línea] disponible en: rua.ua.es/dspace/bitstream/10045/16037/3/ped-09_10-tema3_1.pdf, accedido el 17 de julio de 2012.

RUA. Institutional Repository of the University of Alicante (2012) "Árboles N-arios de Búsqueda". [en línea], disponible en: http://webdiis.unizar.es/asignaturas/EDA-Tardes/árboles_N-arios_de_busqueda.pdf, accedido el 17 de julio de 2012.

Sama Villanueva, S. (2012) "Operaciones básicas de los árboles binarios de búsqueda", [en línea], disponible en: http://www.hci.uniovi.es/Products/DSTool/busqueda/busqueda-operaciones.html, accedido el 16 de julio de 2012.

Schaeffer, E. (2012) *"Árboles B"*. [en línea] disponible en: http://elisa.dyndns-web.com/~elisa/teaching/aa/pdf/clase0410.pdf, accedido el 17 de julio de 2012.

Silva Bijit, L. (2008) “Estructuras de Datos y Algoritmos” Capítulos 12 y 13. [en línea] disponible en: http://www2.elo.utfsm.cl/~lsb/elo320/clases/c12.pdf, accedido el 17 de julio de 2012.

Solar, M. y Figueros, M. (2010) Estructura de datos: Árboles B, Árboles B+ y Árboles B*. [en línea] disponible en: http://www.ramos.utfsm.cl/doc/860/sc/ED-Cap55BTrees12010.pdf, accedido el 17 de julio de 2012.

Soler, Y. y Lezcano, M. (2010). “Organización del conocimiento de la asignatura "Estructuras de Datos y algoritmos" para ingeniería informática basada en mapas conceptuales”. Revista Avances en Sistemas e Informáticas, Vol. 5, No.3, Diciembre, Medellín.

Titiosky, R.S. () Unidad 8: Árboles B, [en línea], disponible en: http://www.ucema.edu.ar/u/rst/Algoritmos_y_Estructura_de_Datos/Teoria/6._Árboles_B.pdf, accedido el 17 de julio de 2012.

UVA (2012) “Definición de un árbol Rojinegro”, [en línea] disponible en: http://www.infor.uva.es/~cvaca/asigs/doceda/rojonegro.pdf, accedido el 17 de julio de 2012.

Universidad de Valladolid (2006)“Introducción a los tipos abstractos de datos”. [en línea] disponible en: http://www.infor.uva.es/~cvaca/asigs/transp0607_4.pdf, accedido el 17 de julio de 2012.

Anexo 1. Diagrama de clases completo del paquete Tree.