UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título : HEA – Herramienta de Software para la enseñanza de árboles B
Autores : Nucilli, Emanuel Alberto
Director : Thomas, Pablo
Codirector : Bertone, Rodolfo
Carrera : Licenciatura en Sistemas

Resumen

En el proceso de enseñanza la motivación juega un papel fundamental. En este sentido, es importante contar con herramientas que ayuden a desarrollar y mejorar este proceso.
Desde la cátedra Introducción a las Bases de Datos de la Facultad de Informática, se planteó la necesidad de contar con una herramienta de software que sirva de complemento en el proceso de enseñanza-aprendizaje, para un tema de relevancia que se dicta en la asignatura. Este tema es el indizado de archivos y el particular uso de estructuras de datos denominadas árboles (particularmente la familia de árboles B), como medio de almacenamiento de índices de archivos.
HEA (Herramienta de Software para la enseñanza de árboles B) permite gestionar a través de animaciones gráficas, la evolución de un árbol B, B* o B+, con operaciones de inserción, eliminación, y/o búsqueda de claves que se encuentran almacenadas en el árbol.
HEA es una herramienta WEB desarrollada íntegramente con tecnología Client Side, permitiendo de esta manera accesibilidad a través de internet, o permitiendo su descarga y utilización de manera local.

Palabras Claves

Árboles, familia de árboles B, árbol B, árbol B+, árbol B, Base de Datos, índices, archivos de datos, tecnología aplicada a la educación, animaciones gráficas.*

Conclusiones

Se ha desarrollado una herramienta de software con fines educativos que presenta particularidades de funcionamiento y de interface, que de acuerdo a la investigación realizada a lo largo de este trabajo no están disponibles.
El proceso de enseñanza-aprendizaje propuesto por Introducción a las Bases de Datos se verá favorecido a partir de brindar al alumno esta herramienta.
HEA ha sido testeado por un grupo de docentes de la cátedra Introducción a las Bases de Datos lográndose su aceptación en cuanto a usabilidad y didáctica.

Trabajos Realizados

Este trabajo se ha enfocado en el diseño y construcción de una herramienta WEB donde la animación de operatoria sobre árboles B, B y B+, ha sido el punto central. El desarrollo de esta herramienta se ha basado en la investigación y utilización de tecnologías adecuadas, que permitan animar gráficamente la evolución de las mencionadas estructuras de datos. Se ha logrado un entorno de aprendizaje fácil de utilizar y comprender, permitiendo que el alumno pueda incorporar fácilmente el crecimiento y decrecimiento de los árboles mencionados, y cómo se realizan las búsquedas de claves en dichas estructuras.*

Trabajos Futuros

A corto plazo, está previsto:

- ✓ Agregar la posibilidad de generar un archivo de carga.
- ✓ Incorporar audio en el historial de operaciones.
- ✓ Incorporar la posibilidad de guardar una sesión de aprendizaje.
- ✓ Generar la posibilidad de impresión.

Fecha de la presentación: Julio de 2011

Índice

Organización de la tesina

Esta tesina está organizada en cuatro partes. En la primera (capítulo 1), se describen cuales fueron los motivos que hicieron que se llevara a cabo el desarrollo de HEA como software que sirva de complemento para la enseñanza de árboles y cuáles son los objetivos a cumplir en la realización de este proyecto.

La segunda parte (capítulo 2, 3 y 4) intenta poner al lector dentro del contexto en el cual trabaja la herramienta. Para esto se introduce el tema de Bases de Datos, su importancia en el manejo y recuperación de información, y especial énfasis en el estudio de la performance en la gestión de datos, mediante la utilización de índices para lograr eficiencia. Dentro de esta parte también se describe el funcionamiento, organización y evolución de las estructuras de datos llamadas árboles utilizadas para la gestión de los índices.

La tercera parte (capítulos 5 y 6) se enfoca en la construcción y funcionamiento de HEA. En ésta se describen aspectos funcionales, visuales y técnicos que ayudan a la utilización y comprensión de la herramienta e informan acerca de la tecnología utilizada para su construcción.

Por último, la parte cuatro plantea en el capítulo 7 las conclusiones alcanzadas a partir de la realización de este proyecto y se mencionan aspectos a mejorar en versiones futuras de la herramienta.

Capítulo 1 – Introducción.

1.1 Motivación

“La palabra motivación proviene de los términos latinos *motus* (“movido”) y *motio* (“movimiento”). Para la psicología y la filosofía, la motivación son aquellas cosas que impulsan a una persona a realizar determinadas acciones y a persistir en ellas hasta el cumplimiento de sus objetivos. El concepto también se encuentra vinculado a la voluntad y al interés. En otras palabras, la motivación es la voluntad para hacer un esfuerzo y alcanzar ciertas metas”. [1]

“La motivación implica la existencia de alguna necesidad, ya sea absoluta, relativa, de placer o de lujo. Cuando una persona está motivada a “algo”, considera que ese “algo” es necesario o conveniente. Por lo tanto, la motivación es el lazo que lleva esa acción a satisfacer la necesidad”. [1]

Transmitir conocimientos es una experiencia altamente gratificante. La motivación no solo del alumno, sino también del docente, juega un papel fundamental. Es importante contar con herramientas que ayuden a desarrollar y mejorar el ámbito educativo. A partir de la definición de motivación se puede observar que términos como *necesario* o *conveniente* tienen una relevancia considerable.

El análisis de docentes de la cátedra Introducción a las Bases de Datos de la Facultad de Informática, concluyó en la necesidad de contar con una herramienta de software que sirva de complemento, tanto para la enseñanza como para el aprendizaje, de un importante tema dictado en la mencionada asignatura. Este tema tiene que ver con estructuras de datos denominadas árboles (particularmente la familia de árboles b), cuyo funcionamiento se explicará más adelante.

Llegar a la conclusión de la conveniencia y necesidad de contar con un complemento educativo, derivó en una primera instancia, en la investigación y búsqueda de un software ya existente que cumpliera tales objetivos.

Esta herramienta de software debía cumplir con pautas precisas a fin de facilitar el aprendizaje del alumno.

La búsqueda resultó infructuosa, debido a que las herramientas encontradas (Slady: Java B-Tree applet animation [2], Animación de método de Borrado [3], Animación del método de inserción [4], Animación del método de búsqueda [5] y simulador de la Universidad de La Serena [6]):

- ✓ No permiten al alumno seleccionar el tipo de árbol de la familia b (b, b+ o b*) con el que desea trabajar.
- ✓ No permiten seleccionar el orden del árbol.
- ✓ No poseen fácil accesibilidad. La utilización de las mismas es a través de internet, sin permitir la descarga para su utilización de manera local.
- ✓ No cuentan con un historial de operaciones que guíe al alumno.

No encontrar una herramienta de software para complementar la enseñanza de árboles, propició la idea de iniciar un desarrollo propio que se adaptase a las necesidades de la asignatura Introducción a las Bases de Datos. Es así que nace el proyecto HEA (Herramienta de Software para la enseñanza de árboles b).

Otros de los motivos que derivaron en la construcción de HEA fue la idea de aplicar tecnología a la educación. El avance tecnológico ha llegado a todos los aspectos de la vida cotidiana, y la educación no es la excepción.

Una de las funciones de la Universidad, es: “Desarrollar el conocimiento en el más alto nivel con sentido crítico, creativo e interdisciplinario, estimulando la permanente búsqueda de la verdad”. [7]

Seguir este precepto, plantea un desafío no solo para el docente y el alumno, sino también para los responsables del desarrollo de tecnologías utilizadas en el proceso de enseñanza-aprendizaje.

Si bien las herramientas de software son de gran ayuda a la hora de trasmitir e incorporar conocimientos, resulta importante no perder de vista, que una de las

características más valiosas, no solo del alumno, sino del ser humano en general, es el espíritu crítico y las ansias de investigación y progreso. Respetar el límite entre brindar herramientas que ayuden en el aprendizaje y que dichas herramientas no apaguen el razonamiento, es un punto fundamental.

La realización de un complemento educativo, que haga más fácil y entretenido el proceso de aprendizaje, sin dejar de lado que dentro de la fase de enseñanza el alumno razone, critique, se pregunte y llegue a conclusiones, es uno de los desafíos que motivaron el desarrollo de HEA.

1.2 Objetivos

Según se menciona en la definición de motivación en el apartado anterior, ésta hace que la persona se mantenga o permanezca realizando una tarea hasta el logro de sus objetivos. Por lo tanto los términos motivación y objetivos tienen una relación muy estrecha.

En términos generales se puede decir que la meta en la construcción de un software académico, es actuar de complemento, para hacer más fácil el proceso de enseñanza-aprendizaje. Ahora bien, es importante aclarar por qué se dice que HEA es un complemento. El objetivo de esta herramienta no es que el alumno que interactúe con ella aprenda los conocimientos de árboles sin haber adquirido previamente conceptos básicos sobre el tema. Es decir que la finalidad de HEA es acompañar al docente en la transmisión de conocimientos y al alumno en su incorporación, basándose en el concepto de que el mejor docente no es el que da las mejores respuestas a las preguntas de sus alumnos sino el que les ayuda a encontrarlas.

Otro de los propósitos de este proyecto es aportar una herramienta a la cátedra de Introducción a las Bases de Datos de la Facultad de Informática. Es importante la concientización del alto nivel de las Universidades Nacionales y el papel fundamental que las mismas tienen en la formación de futuros profesionales que engrandecen a nuestro país y sin dejar de lado el acceso

gratuito a las mismas. Es por esto que en la concepción de este proyecto, la voluntad de aportar humildemente un producto que sirva a la educación, es parte de los objetivos a cumplir.

El tercer objetivo tiene que ver con plasmar lo más fielmente posible los requerimientos de los alumnos. Dicho con otras palabras se intenta que HEA cuente con todos aquellos aspectos visuales y funcionales necesarios, según los propios requerimientos indicados por los alumnos de la cátedra Introducción a las Bases de Datos.

Capítulo 2 – Introducción a las Bases de Datos.

2.1 Organización de una Base de Datos

Antes de conocer la organización de una Base de Datos, es necesario definir el concepto de Base de Datos.

Una definición sencilla indica que una Base de Datos es un conjunto de datos almacenados para su posterior recuperación. A partir de esta definición podemos observar que el concepto de Base de Datos, excede a la informática como disciplina. Cuando se habla, por ejemplo, de una biblioteca, es posible decir que la misma es una Base de Datos, siendo los datos, los libros, revistas, publicaciones, etc.

Hasta aquí nada se ha mencionado acerca de los datos que se almacenan. La pregunta que surge es la siguiente: ¿los datos almacenados guardan algún tipo de relación entre sí? La respuesta es afirmativa, los datos que se guardan en una Base de Datos deben relacionarse. Por lo tanto es posible ampliar la definición original de Base de Datos por la siguiente: conjunto de datos, pertenecientes a un mismo contexto, almacenados para su posterior recuperación.

Almacenar datos para su posterior recuperación es el objetivo primordial de una Base de Datos. Es importante aclarar que estos datos intentan reflejar o representar un problema del mundo real. Por lo tanto el almacenamiento de los datos, no es trivial, sino por el contrario, requiere de un análisis previo que permita abstraer el problema real y modelar correctamente los datos necesarios para su resolución.

El modelo de datos es un elemento fundamental en una Base de Datos. El modelo se define como un conjunto de herramientas conceptuales que permiten describir los datos y su semántica, sus relaciones y las restricciones de integridad y consistencia [8].

Existen grupos de modelos de datos:

- ✓ Modelos lógicos basados en objetos
- ✓ Modelos lógicos basados en registros

2.1.1 - Modelos lógicos basados en objetos

Estos modelos son utilizados para representar la información de un problema del mundo real con un esquema de alto nivel de abstracción. Dentro de estos modelos se destacan el modelo de entidades y relaciones, y el modelo orientado a objetos.
El modelo de datos entidad-relación (E-R) está integrado por entidades y sus relaciones.
Se define entidad a un objeto de la realidad que se describe por un conjunto de características denominados atributos. Actualmente este modelo es el más utilizado debido a su simplicidad y al mismo tiempo a su potencia expresiva.
El modelo orientado a objetos representa la realidad con objetos de similares características a los objetos del mundo real.

2.1.2 - Modelos lógicos basados en registros

Estos modelos utilizan la estructura de datos de registro para almacenar la información en una Base de Datos. Existen tres modelos basados en registros: modelo jerárquico, modelo de red y modelo relacional.
La esencia del modelo jerárquico consiste en utilizar registros vinculados entre sí formando una estructura de árbol.
El modelo en red también utiliza registros con vínculos que generan una estructura de grafo.
El modelo relacional utiliza un conjunto de tablas para representar la información. Una tabla se compone por filas, denominadas tuplas, y columnas. Cada fila representa un registro y cada columna un campo o atributo. Estas

tablas se vinculan entre si y establecen relaciones de integridad entre los datos que la componen. Este modelo es el más utilizado desde hace tres décadas.

Cuando una herramienta de software trabaja con datos, estos se encuentran en memoria RAM. La información contenida en este tipo de memoria, se dice que es volátil, es decir, que cuando la computadora se apaga, estos datos se pierden. El problema surge cuando es necesario persistir la información. En la mayoría de los casos se desea que los datos perduren en el tiempo para futuros accesos y consultas. Para ello se utilizan los archivos. Los archivos se guardan en dispositivos que constituyen la denominada memoria secundaria. Ejemplo de este tipo de dispositivos son los discos rígidos, Cd, Dvd, pen drive, entre otros. Todos ellos permiten que los datos persistan a lo largo del tiempo, aún cuando un programa que los utiliza ha finalizado su ejecución o cuando la computadora se apaga.

Existen dos visiones sobre un archivo. La primera es una visión física, que tiene que ver con el almacenamiento real del archivo en memoria secundaria, y la segunda visión es la que se denomina visión lógica basada en como un programa ve al archivo físico. Cuando un programa necesita trabajar con un archivo, las referencias a este son lógicas. El programa trabaja con su propio conjunto de direcciones, que luego el sistema operativo se encargará de trasformar para traducir estas direcciones lógicas a direcciones físicas que indicarán el lugar exacto del archivo en memoria secundaria.

En definitiva, se puede afirmar, que una Base de Datos es un conjunto de archivos almacenados en memoria secundaria. Para una mayor comprensión podemos ejemplificar la siguiente Base de Datos de compras.

ARCHIVO DE PRODUCTOS

Código artículo	Descripción del material	Unidad	Cantidad
1.01.01	CD-ROM RW IDE	Unidad	10
1.01.02	Disco rígido ATA 66	Unidad	20
1.02.01	Disco Flexible de 3 1/2" 1,44 Mbytes	Caja de 10	20
2.01.01	Sonido de 16 bit	Unidad	5
3.01.01	Papel carta para impresora.	Resma 100 hojas	25
4.01.01	Pentium II 200Mhz	Unidad	7
4.01.02	Pentium III 500Mhz	Unidad	8
4.01.03	Pentium III 800Mhz	Unidad	9

ARCHIVO DE PROVEEDORES

Código proveedor	Nombre del proveedor	Teléfono del proveedor	Dirección del proveedor
001	Inca Tel	4923-4803	Av. La Plata 365
002	Infocad	4633-2520	Doblas 1578
003	Herrera Compusistem	4232-7711	Av. Rivadavia 3558

ARCHIVO DE ORIGEN DE LOS PRODUCTOS

Código proveedor	Código del artículo	Precio
001	1.01.01	70,00
002	1.01.01	80,00
003	1.01.01	75,00
002	2.01.01	50
001	4.01.03	450

Esta Base de Datos contiene información de tres Entidades:

- ✓ Datos sobre productos (Entidad producto), almacenados en el archivo de Productos;
- ✓ Datos sobre proveedores (Entidad proveedores), almacenados en el archivo Proveedores y;
- ✓ Datos sobre el origen de los productos (Entidad origen del producto), o sea, los productos son provistos por cada proveedor y viceversa, almacenados en el archivo de Origen del producto.

La información almacenada en cada uno de estos archivos es lo que denominamos entidad. Por lo tanto una entidad es cualquier persona, cosa o evento, real o imaginario, de interés para la organización y acerca del cual se capturan, almacenan o procesan datos.

Además, cada uno de estos archivos está formado por un conjunto de registros que describe, a través de los atributos o datos (columna), cada entidad en él almacenado.

Todos los registros de un archivo, identificados por las filas de cada tabla, poseen el mismo formato, o sea tienen el mismo conjunto de datos o atributos, identificados por las columnas, que describen a las entidades.

En otras palabras los registros están formados por un conjunto de datos almacenados en los campos de cada atributo; y cada registro debe contener el conjunto de atributos necesarios, para describir completamente cada entidad sobre la cual una organización necesita almacenar y obtener información [9].

Hasta aquí se ha descripto la organización de una Base de Datos, afirmando que la información que deseamos persistir debe almacenarse en memoria secundaria.

Es importante remarcar dos diferencias fundamentales entre la memoria RAM y la memoria secundaria:

- ✓ Capacidad de almacenamiento: el volumen de información que se puede almacenar en la RAM es considerablemente menor al de la memoria secundaria.
- ✓ Velocidad de acceso: el acceso a datos que se encuentran en la memoria RAM es mucho más rápido que el acceso a datos contenidos en memoria secundaria. Al hablar de acceso a datos en la RAM, se mide en término de nanosegundos (10^{-9} segundos) mientras que el acceso a memoria secundaria se mide en término de milisegundos (10^{-3} segundos).

La segunda diferencia resulta fundamental ya que en el caso de querer persistir información, como se mencionara oportunamente, se hará en la memoria secundaria; por lo tanto, es importante la utilización de técnicas que mejoren la eficiencia en el momento de la recuperación de los datos.

Una de las técnicas utilizadas para mejorar la eficiencia en la recuperación de datos es el manejo de índices, tema de estudio del próximo capítulo.

Capítulo 3 - Índices.

Las operaciones posibles sobre una Base de Datos son la inserción, la eliminación y la búsqueda de información. Dentro de estas operaciones, el 80% son operaciones de consulta, razón por la cual, resulta fundamental, reducir lo máximo posible el tiempo de búsqueda de información sobre una Base de Datos.

Lo primero que se piensa al tener que lograr velocidad de acceso es que la información este contenida en memoria RAM.

Debido a que las Bases de Datos pueden almacenar gran cantidad de información y sabiendo que una de las limitaciones de la memoria RAM es su capacidad de almacenamiento, se debe pensar como alternativa el almacenamiento de la información en memoria secundaria. Por otro lado se desea que la información contenida en una Base de Datos persista, por lo tanto, conociendo la volatilidad de almacenamiento de la memoria RAM, no quedan dudas que será necesario utilizar la memoria secundaria como medio de almacenamiento. La información en memoria secundaria se almacena a través de archivos.

Según las propiedades de la memoria secundaria, vistas al final del capítulo anterior, el acceso a la información es costoso, por lo tanto, será necesario la utilización de diferentes técnicas en pos de subsanar este problema y lograr la mayor eficiencia posible en la recuperación de información.

El caso más simple consiste en disponer de un archivo serie, es decir, sin ningún orden preestablecido más que el físico, donde para acceder a un registro determinado, se deben visitar todos los registros previos en el orden que estos fueron almacenados. Aquí, la búsqueda de un dato específico se detiene en el registro que contiene ese dato (previo acceso a todos los registros anteriores), o en el final del archivo si el resultado no es exitoso (el dato no se encuentra). Por lo tanto el mejor caso es ubicar el dato deseado en el primer registro (1 lectura), y el peor caso en el último registro (N lecturas,

siendo N la cantidad de registros). El caso promedio consiste entonces en realizar N/2 lecturas. Es decir, la performance depende de la cantidad de registros del archivo, siendo entonces de orden N.

Si el archivo estuviese físicamente ordenado y el argumento de búsqueda coincidiera con el criterio de ordenación, la variación en relación al caso anterior se produce para el caso que el dato buscado no se encuentre en dicho archivo. En ese caso, el proceso de búsqueda se detiene en el registro cuyo dato es “mayor” al buscado. De ese modo, en ese caso no existe la necesidad de recorrer todo el archivo. No obstante, la performance sigue siendo de orden N, y a la estrategia de búsqueda utilizada en ambos casos se la denomina secuencial.

Si se dispone de un archivo con registros de longitud fija y además físicamente ordenado, es posible mejorar esta performance de acceso si la búsqueda se realiza con el mismo argumento que el utilizado para ordenar este archivo. En este caso, la estrategia se ve alterada. La primera comparación del dato que se pretende localizar es contra el registro medio del archivo, es decir el que tiene NRR=N/2. Cada registro tiene un número relativo dentro del archivo. Dicho número se denomina NRR y permite calcular la distancia en bytes desde el principio del archivo hasta cualquier registro.

Si el registro no contiene ese dato se descarta la mitad mayor o menor, según corresponda, reduciendo el espacio de búsqueda a los N/2 registros restantes (mitad restante). Nuevamente se realiza la comparación pero con el registro medio de la mitad restante, repitiendo así este proceso hasta reducir el espacio de búsqueda a un registro.

En el ejemplo siguiente se busca el elemento 4, como el archivo contienen 20 registros el primer NNR accedido es el registro del décimo lugar.

1 3 4 5 8 11 15 18 21 25 29 30 40 43 45 56 67 87 88 90

Como en esa posición está el elemento 25 los registros posteriores al decimo se descartan.

Se calcula el nuevo punto medio, que será la posición 4 del archivo.

Como en esta posición tampoco está el elemento buscado se descartan los mayores al 5.

El proceso calcula nuevamente el punto medio y descarta los menores.

En la próxima pasada, elemento 4 es encontrado.

Este criterio de búsqueda, donde la mitad de los registros restantes se descartan en cada comparación se denomina búsqueda binaria.

En la búsqueda binaria, si N es el número de registros almacenados en un archivo, el orden de búsqueda será de $Log_2(N)$, teniendo en cuenta los accesos. Se puede notar que utilizando esta técnica, se mejora el orden de búsqueda con respecto a la búsqueda secuencial la cual tenía un orden de búsqueda lineal.

El principal inconveniente de la búsqueda binaria es la necesidad de que el archivo este físicamente ordenado. El proceso de ordenación es un proceso costoso que aumenta la complejidad. No es objetivo de este apartado describir las diferentes técnicas de ordenación que existen, pero si es oportuno nombrar que el ordenamiento de los elementos de un archivo posee un estudio particular y existen diferentes técnicas para llevar a cabo esta tarea.

Si bien la búsqueda binaria mejora la performance con respecto a la secuencial, su costo de implementación sigue siendo demasiado alto, es por

ello que se plantea una nueva alternativa. La idea consiste en poder manejar un archivo como si estuviese ordenado sin que realmente lo esté. Dicho de otra forma, simular el ordenamiento de un archivo, salvando de esta manera el costo que conlleva. Para poder llevar a cabo esta idea es que se piensa en la utilización de índices.

La mayoría de los libros poseen un índice, el cual es una tabla que contiene un conjunto de temas y un número de página donde puede encontrarse dicho tema. A la lista de temas que contiene un índice se los denomina claves o llaves, y los números de páginas son los campos de referencia. Todos los índices están basados es este concepto básico de claves o llaves y campos de referencia.

El índice de un libro permite localizar información rápidamente. La utilización del índice mejora notablemente la performance de búsqueda en comparación a una búsqueda hoja por hoja o secuencial. Por lo tanto un índice es un recurso para buscar información.

Una de las ventajas que posee la utilización de índices se basa en la indirección. Debido a que los índices trabajan con indirecciones, es posible buscar información en un archivo de datos que no necesariamente debe estar físicamente ordenado, permitiendo de esta manera, disminuir el costo adicional que conlleva la ordenación de los datos en un archivo.

Por otro lado, otra de las ventajas que proporciona la utilización de índices es la posibilidad de tener varios caminos de búsqueda. Para aclarar este concepto se presenta un ejemplo que consiste en ordenar los libros de una biblioteca, de manera tal de poder localizarlos rápidamente por autor, titulo y editorial. Una primera alternativa sería tener tres copias de cada libro y tres edificios. En cada edificio se ordena una de las copias de cada libro por uno de los criterios mencionados anteriormente.

Es indudable que este método es altamente costoso. Sin embargo, a través de la utilización de índices se podría tener un catalogo de tarjetas que sea un conjunto de tres índices, donde cada uno tiene un campo clave distinto pero

todos tienen el mismo número de catalogo como campo de referencia. Por lo tanto, el uso de la indización permite tener varios caminos de acceso a un archivo.

Cuando se realiza la búsqueda de un dato se debe considerar la cantidad de accesos a disco en pos de encontrar esa información, y en cada acceso la verificación si el dato obtenido es el buscado (comparación). Es así que surgen dos parámetros a analizar: cantidad de accesos y cantidad de comparaciones. El primero de ellos es una operación sobre memoria secundaria, lo que implica un costo relativamente alto; mientras que el segundo es sobre memoria principal, por lo que el costo es relativamente bajo. A modo de ejemplo, si un acceso a memoria principal representa 5 segundos, un acceso a memoria secundaria representaría 29 días. Por lo tanto, el acceso a memoria secundaria es sobre el que se posa la atención en función de mejorar los tiempos de acceso.

Para comprender mejor el funcionamiento de los índices se presenta el siguiente archivo de datos a modo de ejemplo:

Dir. Reg.	Cía.	Código	Título	Grupo Musical Compositor	Interprete
15	AGP	23	El final es en donde partir.	La Renga	La Renga
36	CAT	13	Solo le pido a Dios	León Gieco	León Gieco
83	EMI	11	Alternativo	Babasonicos	Babasonico
118	CAT	15	Palomar	Los Piojos	Los Piojos
161	MAG	2310	Despedida	Daddy Yankee	Dady Yankee
209	GPS	1323	Tu sin Mi	Dread Mar I	Dread Mar I

111	SON	22	Esquivando Charcos	La Renga	La Renga

Al crearse el archivo, con clave primaria formada por los atributos Cía. + Código, se crea el índice primario asociado. El índice y el archivo quedarían de la siguiente manera:

Clave	Ref.
AGP23	15
CAT13	36
CAT15	118
EMI11	83
GPS1323	209
SON22	111

Dir. Reg.	Registro de Datos
15	AGP|23|El final es en donde partir |La Renga |La renga
36	CAT|13|Solo le pido a Dios |Leon Gieco |Leon Gieco
118	CAT|15|Palomar|Los Piojos |Los Piojos
83	EMI|11|Alternativo |Babasonicos |Babasonico
209	GPS|1323|Tu sin mi |Dread Mar I |Dread Mar I
111	SON|22|Esquivando charcos |La Renga |La Renga

La tabla de la izquierda presenta el índice generado ordenado por la clave primaria del archivo de datos, en tanto que la tabla de la derecha muestra el archivo de datos propiamente dicho. Es de notar que el archivo de datos no está ordenado.

Para realizar cualquier búsqueda de datos, por ejemplo si se desea buscar el compositor del tema “Palomar”, en primer lugar se busca la clave primaria (CAT15) en el índice. Una vez hallada la referencia (118), correspondiente a la dirección del primer byte del registro buscado, se accede directamente al archivo de datos según lo indicado por dicha referencia.

3.1 - Índices Secundarios

La pregunta que seguramente le surge al lector al buscar el compositor de la canción “Palomar” por la clave primaria (CAT15) es cómo saber este dato, la clave primaria. No es natural ni intuitivo solicitar un dato por clave primaria, sino por el nombre de la canción o eventualmente autor, que son atributos mucho más fáciles de recordar. Estos atributos, nombre de canción o autor, podrían contener valores repetidos en el archivo original. Por este motivo no es posible pensarlos como parte de una clave primaria. La clave que soporta valores repetidos se denomina clave secundaria.

Por lo tanto, es necesario crear otro tipo de índice mediante el cual se pueda acceder a la información de un archivo, pero con datos fáciles de recordar. De esta manera surge el uso de índices secundarios.

Un índice secundario es una estructura adicional que permite relacionar una clave secundaria con una o más claves primarias, dado que como ya se ha mencionado previamente, varios registros pueden contener la misma clave secundaria. Luego para acceder a un dato deseado, primero se accede al índice secundario por clave secundaria, allí se obtiene la clave primaria, se accede con dicha clave al índice primario para obtener finalmente la dirección efectiva del registro que contiene la información buscada. La pregunta que podría surgir es por qué motivo el índice secundario no posee directamente la dirección física de elemento de dato. La respuesta es simple, al tener la dirección física solamente definida para la clave primaria, si el registro cambia de lugar en el archivo, solo debe actualizarse la clave primaria. Suponga que para un archivo cualquiera se tienen definido un índice primario y cuatro secundarios. Si se modifica la posición física de un registro en el archivo de datos, los cinco índices deberían modificarse. Para mejorar esta situación, solo el índice primario tiene la dirección física y es el único que debe alterarse, todos los índices secundarios permanecen sin cambios.

Continuando con los datos del ejemplo presentado anteriormente, en la siguiente tabla se muestra un índice secundario (índice de Grupos de Música)

Clave Secundaria	Clave Primaria
Babasonicos	EMI11
Dread Mar I	GPS1323
La Renga	AGP23
La Renga	SON22
Leon Gieco	CAT13
Los Piojos	CAT15

En el ejemplo puede verse que cuando la clave secundaria se repite, por ejemplo en La Renga, existe un segundo criterio ordenación, establecido por la clave primaria.

Si se desea buscar un tema del grupo Babasonicos, primero se busca la clave Babasonicos en el índice secundario, se obtiene allí la clave primaria EMI11 y finalmente se accede al índice primario para obtener luego la dirección física del archivo (83).

Los índices, son una porción reducida de información con respecto a la información total almacenada en los registros de un archivo. Esta información es necesario manejarla de manera adecuada para lograr la mejor performance posible.

El espacio en disco requerido para almacenar el índice es típicamente menor que el espacio de almacenamiento de la tabla (puesto que los índices generalmente contienen solamente los campos clave de acuerdo con los que la tabla será ordenada, y excluyen el resto de los detalles de la tabla). En una base de datos relacional un índice es una copia de parte de una tabla.

Frente a lo expuesto, el diseñador de una Base de Datos se puede ver tentado a la creación excesiva de índices. Ahora bien, es necesario saber que el mantenimiento de los índices conlleva una carga adicional de procesamiento. De manera tal, que cada vez que se haga una nueva inserción o actualización

sobre una tabla de una Base de Datos, los índices también deberán ser actualizados, razón por la cual, se debe lograr el equilibrio justo, para indexar aquellos campos que realmente sean necesarios a la hora de optimizar la recuperación de información.[10]

Si el tamaño de los índices es lo suficientemente pequeño como para caber en memoria RAM, el manejo de los mismos se facilita, debido a que el ordenamiento de los índices en RAM es un proceso poco costoso que permitirá realizar búsquedas binarias accediendo de manera eficiente a los datos de un archivo sin que los elementos del mismos deban estar ordenados.

Lamentablemente la mayoría de las veces los índices no pueden ser manejados en memoria RAM, razón por la cual es necesario almacenarlos en un archivo en memoria secundaria. Sin bien el ordenamiento de un archivo en memoria secundaria es un proceso costoso, el ordenamiento de índices sigue siendo conveniente respecto a la posibilidad de ordenar el archivo de datos, debido a que el tamaño del archivo de índices es menor. Sin embargo el número de desplazamientos en memoria secundaria sigue siendo inaceptable, razón por la cual, es necesario buscar una alternativa que permita la organización correcta de los índices en pos de lograr una búsqueda eficiente.

Esta alternativa consiste en organizar los índices sobre árboles B, B+, B* o sobre una mezcla de ellos. Estas estructuras son sobre las que trabaja HEA y serán analizadas con profundidad en el próximo capítulo.

Capítulo 4 – Árboles

4.1 - Introducción

Como se describió en el capítulo anterior es muy importante la utilización de índices para mejorar la performance en las operaciones de consultas sobre una Base de Datos. Un índice es una porción de información, por lo tanto, la eficiencia en el manejo genera un nuevo desafío.

Uno de los problemas radica en el tamaño que tienen los índices. A medida que el volumen de información a almacenar crece, y sabiendo que el límite de almacenamiento de la memoria RAM es mucho menor que el de la memoria secundaria, pero que los accesos a memoria secundaria son mucho más costosos, se debe resignar la velocidad de acceso en pos de la capacidad de almacenamiento.

El presente capítulo detalla un tipo de estructura no lineal, llamada árbol, que se utiliza para optimizar la organización de los índices, haciendo un análisis evolutivo de dichas estructuras hasta llegar a los árboles B, B+ y B*, árboles sobre los cuales trabaja HEA.

4.2 – Consideraciones generales

Los árboles son estructuras de datos no lineales utilizadas para organizar información. Las estructuras de datos lineales son aquellas en donde a cada elemento le corresponde no más de un elemento siguiente. En las estructuras de datos no lineales, como los árboles, cada elemento puede tener más de un elemento siguiente.

Este tipo de estructuras tienen diferentes aplicaciones, como por ejemplo, el manejo de sistemas de archivos, la jerarquía de clases en lenguajes orientados a objetos, la organización de índices en Bases de Datos entre otras.

La utilización de árboles para la organización de índices de una Base de Datos será motivo de estudio del presente capítulo.

Existen diferentes tipos de árboles, cada uno de los cuales posee características particulares las cuales serán explicadas más adelante:

- ✓ Binarios
- ✓ AVL
- ✓ Multicamino
- ✓ Balanceados (B,B*,B+)

Un árbol se caracteriza por estar formado por un conjunto de nodos conectados por una serie de aristas que verifican que:

- ✓ Hay un único nodo raíz
- ✓ Cada nodo, excepto la raíz, tiene un único padre
- ✓ Hay un único camino (desde la raíz hasta cada nodo)
- ✓ Puede estar ordenado o no

Es importante para la correcta comprensión de este tema, conocer en profundidad la terminología utilizada, representada en la figura 4.2.1:

- ✓ Raíz: único nodo sin padre
- ✓ Nodo intermedio: nodo ubicado entre la raíz y las hojas.
- ✓ Nodo hoja: no tiene hijos.
- ✓ Descendiente directo: hijo.
- ✓ Descendientes: hijo, nieto...
- ✓ Subárbol: árbol formado por un nodo y sus descendientes.

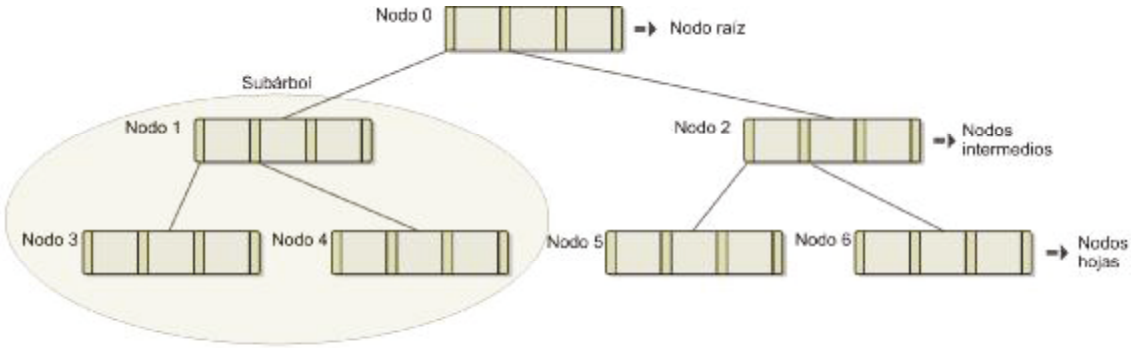

Figura 4.2.1

- ✓ Grado de un nodo: número de descendientes directos
- ✓ Grado del árbol: mayor grado de sus nodos
- ✓ Árbol binario: árbol de grado 2
    - Cada nodo tiene a lo sumo dos descendientes directos.
- ✓ Árbol multicamino:
    - Cada nodo puede tener n descendientes directos
- ✓ Lista : árbol degenerado de grado1
- ✓ Altura de un nodo: número de predecesores (figura 4.2.2)
- ✓ Altura del árbol: profundidad máxima de cualquier nodo (figura 4.2.2)

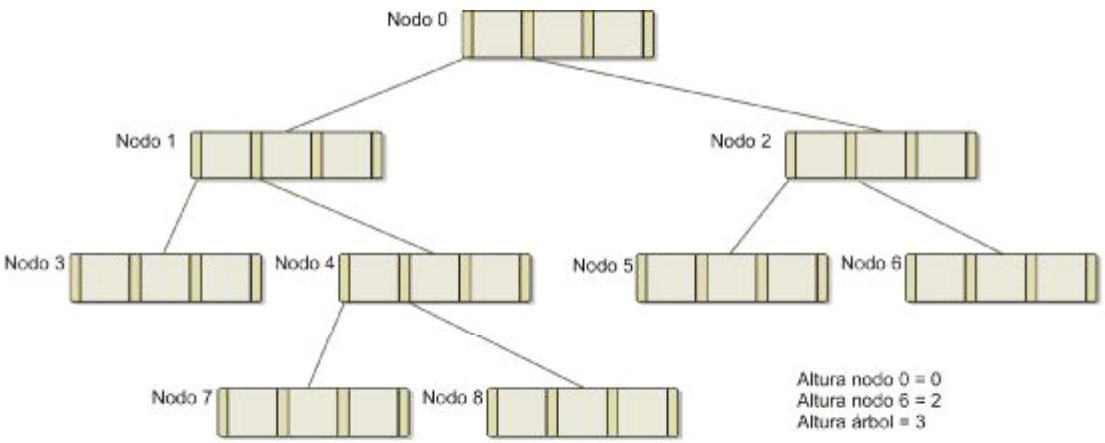

Figura 4.2.2

- ✓ Camino: existe un camino del nodo X al nodo Y, si existe una sucesión de nodos que permitan llegar desde X a Y. (figura 4.2.3)

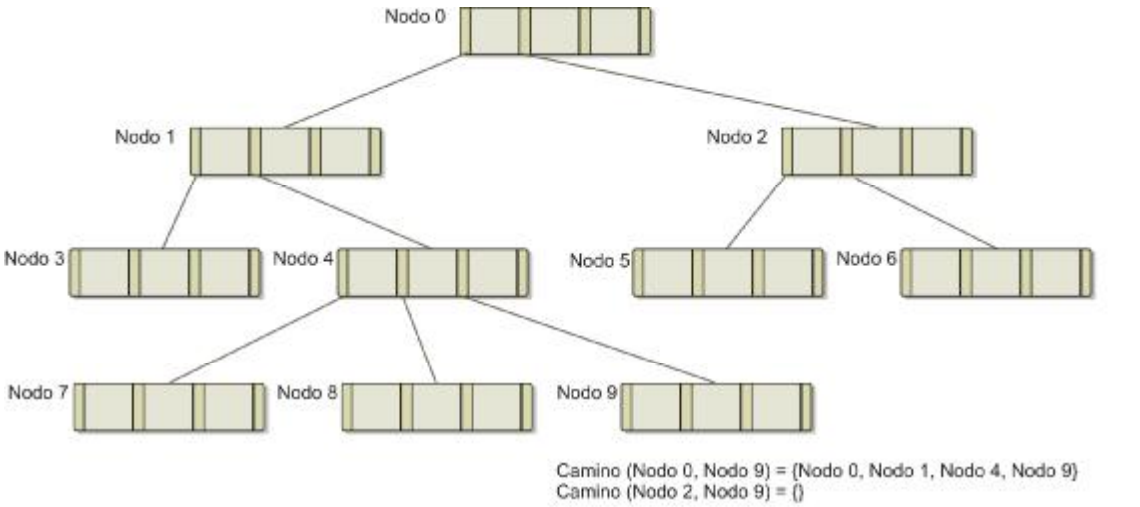

Figura 4.2.3

4.3 – Evolución

4.3.1 - Árboles Binarios

Un árbol binario es una estructura de datos dinámica no lineal, en la cual cada nodo (elementos de dato), puede tener a lo sumo dos hijos [8].

Este tipo de estructura, basa su funcionamiento en el orden de los elementos que almacena, por lo tanto, mantener esta propiedad es una tarea relevante a la hora de insertar un nuevo elemento en el árbol. De esta manera, como se muestra en la figura 4.3.1, los datos son almacenados en orden. Dado un elemento x, los elementos menores a x se encontrarán a su izquierda y los mayores a x a su derecha.

Si los elementos de un árbol binario no se encontrarán ordenados, la búsqueda de información sería menos eficiente debido a que no se podría realizar un tipo

de búsqueda denominada búsqueda binaria, la cual será explicada a continuación.

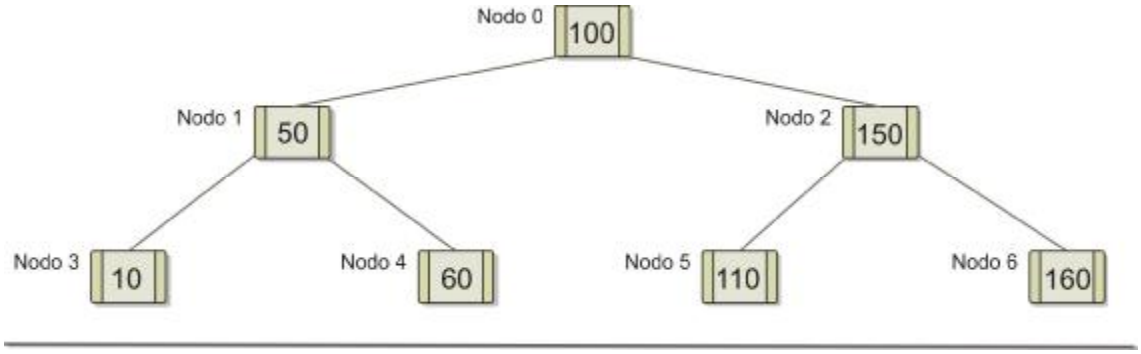

Figura 4.3.1

El orden de los elementos de un árbol binario, permite lo que se denomina búsqueda binaria. Este tipo de búsqueda funciona de la siguiente manera: se realiza en orden descendiente, es decir, se comienza desde la raíz del árbol hacia sus hojas. Se analiza el elemento de la raíz, si es mayor al elemento buscado, se descarta toda rama derecha del árbol, y se repite el procedimiento solo con la rama izquierda. Es por esto que el ordenamiento de los elementos del árbol es fundamental. La búsqueda de información en un árbol binario es de orden logarítmico. Encontrar un elemento es del orden $Log_2(N)$, siendo N la cantidad de elementos distribuidos en el árbol.

Un árbol binario es almacenado en un archivo en memoria secundaria. Si bien el acceso a memoria secundaria es más costoso que a memoria principal, la limitación de espacio de la memoria principal y la necesidad de persistir los datos, han sido dos motivos suficientes para que almacenar un árbol binario en memoria secundaria a través de un archivo sea la mejor decisión.

Una ventaja de la organización mediante árboles binarios está dada en la inserción de nuevos elementos. Mientras que un archivo se desordena cuando se agrega un nuevo dato, si la organización se realiza con la política de árbol binario la operatoria resulta más sencilla en términos de complejidad

computacional. La secuencia de pasos para insertar un nuevo elemento es la siguiente:

1. Agregar el nuevo elemento de dato al final del archivo.
2. Buscar al padre de dicho elemento. Para ello se recorre el archivo desde la raíz hasta llegar a un nodo terminal.
3. Actualizar el padre, haciendo referencia a la dirección del nuevo hijo.

Se puede observar la performance desde el punto de vista de accesos a disco: se deben realizar $Log_2(N)$ lecturas, necesarias para localizar el padre del nuevo elemento, y dos operaciones de escritura (el nuevo elemento y la actualización del padre).

La operación de borrado presenta un análisis similar. Para quitar un elemento de un árbol, éste debe ser necesariamente un elemento terminal. Si no lo fuera, debe intercambiarse el elemento en cuestión con el menor de sus hijos mayores. En la operatoria de borrado, deberán realizarse nuevamente, $Log_2(N)$ lecturas y dos escrituras.

La eficiencia en las operaciones sobre un árbol binario, está supeditada no solo a la ubicación de los elementos contenidos en el árbol sino también al balanceo del mismo. Se dice que un árbol esta balanceado cuando la trayectoria de la raíz a cada una de las hojas está representada por igual cantidad de nodos. Es decir, todos los nodos hoja se encuentran a igual distancia del nodo raíz, tal como se muestra en la figura 4.3.2.

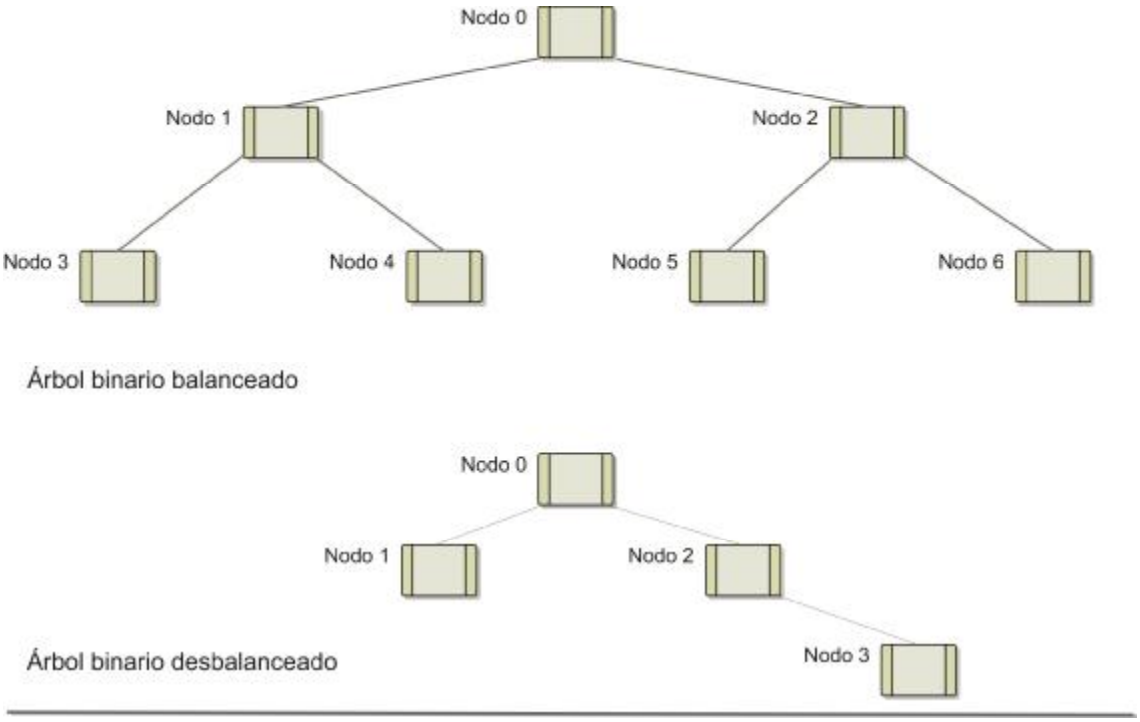

Figura 4.3.2

Este es un gran problema con el que se enfrentan los árboles binarios. Supongamos la siguiente secuencia de inserción sobre un árbol binario: 1, 2, 3, 4, 5. Como se puede observar en la figura 4.3.3, por el funcionamiento de este tipo de árboles, recordemos que los elementos menores a uno dado van a la izquierda del mismo y los mayores a su derecha, la estructura que resulta es una lista encadenada.

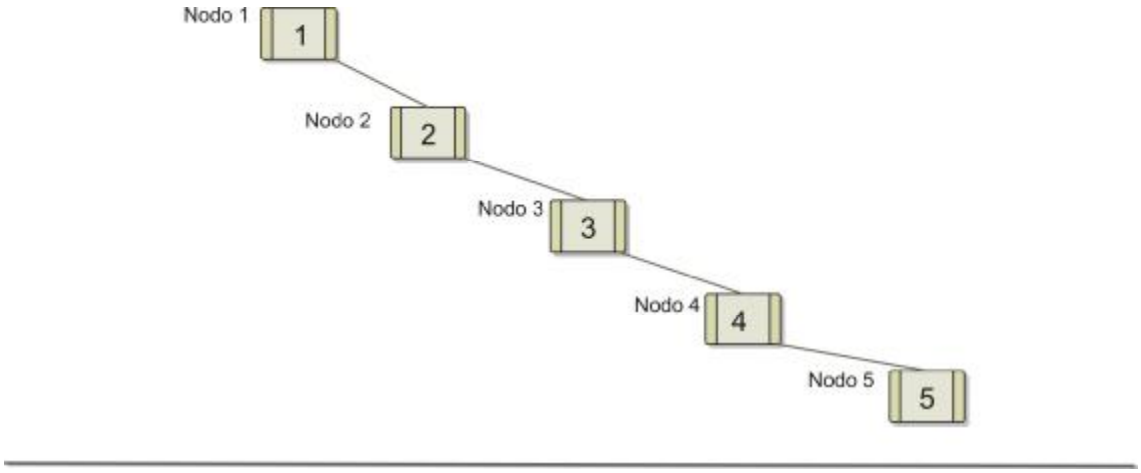

Figura 4.3.3

En este caso los órdenes de búsqueda planteados anteriormente se ven alterados, disminuyendo su performance, y convirtiéndose en un orden N o comúnmente llamado orden lineal.

Claramente la eficiencia de utilizar un árbol binario como estructura de datos para almacenar un índice, va a estar supeditada al balanceo del árbol, que como se mostró previamente no es fácil de lograr.

4.3.2 - Árboles AVL

A partir del problema de balanceo presentado en los árboles binarios es que aparecen los árboles AVL. Este tipo de árboles, también llamados árboles balanceados en altura, son árboles binarios con una característica adicional. La diferencia máxima entre las alturas de dos subárboles cualesquiera que tienen raíz común, no supera un delta determinado, siendo delta lo que se denomina el nivel de balanceo del árbol.

Por lo tanto, si un árbol balanceado tiene un delta 1, la diferencia en altura de dos ramas que tengan un padre común no podrá diferir en más de un nodo.

Bajo estas pautas un árbol AVL con delta 1 está casi balanceado y se podría pensar como una alternativa de estructura de datos para almacenar un índice

Si una secuencia de inserción hace que se viole el valor delta, se debe implementar una operación extra llamada rebalanceo del árbol. Esta operación aumenta considerablemente los costos en las inserciones y borrado de elementos, haciendo que tanto los árboles binarios como los AVL no sean una buena solución a la hora del manejo de índices.

Según [I0] para 1.000.000 claves, un árbol completamente balanceado requiere desplazamiento en 20 niveles para buscar alguna de las claves. En un árbol AVL, el número máximo de niveles a buscar es 28. Esto es bueno para almacenamiento en memoria principal. En almacenamiento secundario es deseable a lo sumo 5 o 6 desplazamientos, 20 o 28 es inaceptable.
Por lo tanto, los árboles AVL no son estructuras de datos adecuadas para almacenar índices.

4.3.3 - Árboles binarios paginados

Cuando se transfiere información desde o hacia el disco rígido dicha transferencia no se limita a un registro, sino que son enviados un conjunto de

registros; es decir, los que quepan en un buffer de memoria. Las operaciones de lectura y escritura de datos de un archivo utilizando buffers presentan una mejora de performance. Este concepto es de utilidad cuando se genera el archivo que contiene el árbol binario. Dicho árbol se puede dividir en páginas, es decir paginarlo, donde cada página contenga un conjunto de nodos, los cuales estén ubicados en direcciones físicas cercanas. Así, cuando se transfieren datos, no se accede al disco para transferir unos pocos bytes, sino que, se transmite una página completa. Una organización de este tipo reduce el número de accesos a disco, necesarios para poder recuperar la información.

Al dividir un árbol binario en páginas es posible realizar búsquedas más rápidas de datos almacenados en memoria secundaria. No obstante, para analizar la performance resultante deberían tenerse en cuenta la cantidad de nodos que caben en una página. Así, suponiendo que en un buffer caben 255 elementos, el tamaño de cada página sería entonces de 255 nodos, resultando la performance final de búsqueda del orden de $\log_{256}$ (N), es decir $\log_{k+1}$ (N), siendo N la cantidad de claves del archivo y K la cantidad de nodos por página. Volviendo al ejemplo aplicado en los árboles AVL, en caso de tener 1.000.000 de claves y suponiendo que cada página sería de 255 nodos, la performance final de búsqueda sería $\log_{256}$ (1.000.000) = 2.55 aproximadamente, resultado que se encuentra dentro de los limites deseables para el manejo de índices.

La cuestión a analizar, ahora, es como generar un árbol binario paginado. Para que el uso de las páginas tenga sentido, los elementos correspondientes a una misma página deberían llegar consecutivos para almacenarse en el mismo sector de disco. En su defecto, sería imposible que una página contuviese los elementos relacionados. Como esta opción es imposible, no se puede condicionar la llegada de elementos al archivo de datos, es menester encontrar otra solución. Así, cada página deberá quedar incompleta si al llegar un elemento no estuviese relacionado con los anteriores. Es decir, un elemento deberá estar contenido en la página que le corresponde, en lugar de quedar en la primera disponible.

Para lograr esta construcción hay que pensar en páginas, los elementos que caben en cada una de ellas, la forma en que se construye un árbol binario y, además, los temas relacionados con el balanceo.

Dividir el árbol en páginas implica un costo extra necesario para su reacomodamiento y para mantener su balanceo interno. Un algoritmo que soporte esta construcción será muy costoso de implementar y además muy costoso en términos de performance.

Aquí se plantea una disyuntiva, el paginado de árboles binarios con su beneficio y el costo que ello trae aparejado. La solución para este problema consiste en adoptar la idea de manipular más de un registro (es decir, la página) y tratar de disponer de algoritmos a bajo costo para construir un árbol balanceado. Esta idea es motivadora principal al uso de otro tipo de árboles, los utilizados por HEA, tal como se mostrará más adelante en esta Tesina.

4.3.4 - Árboles Multicamino

Los árboles multicamino nacen a partir de la idea de generalizar el concepto de árboles binarios. Como se indicó en los apartados anteriores, un árbol binario, es aquel en el que cada nodo del árbol puede tener a lo sumo dos nodos hijos o descendientes.

A partir de este concepto es posible afirmar que un árbol ternario, sería entonces aquel donde cada nodo posee dos elementos y tres descendientes o hijos. La figura 4.3.4 muestra un ejemplo de un árbol ternario.

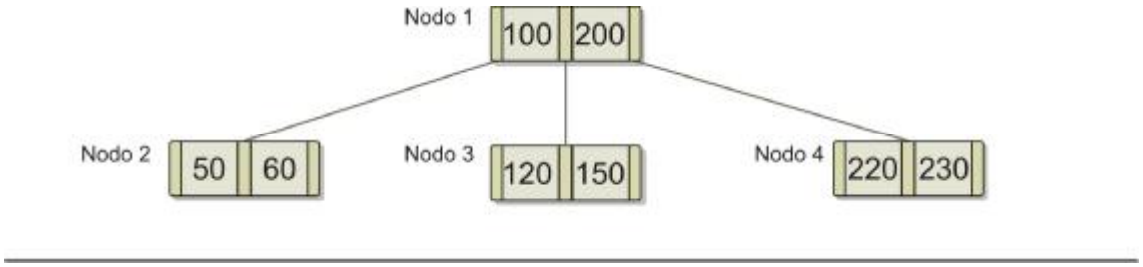

Figura 4.3.4

Ahora bien, es posible generalizar este concepto, a partir de un nuevo parámetro llamado orden del árbol. De este modo un árbol con orden N, será aquel donde cada nodo pueda contener a lo sumo N-1 elementos y a los sumo N descendientes o hijos. Un árbol con estas características se denomina árbol multicamino o árbol n-ario.

Sin embargo, el problema de desbalanceo planteado para los árboles binarios sigue latente. Es motivo de análisis del próximo apartado, la solución al problema de desbalanceo de árboles.

4.4 – Árboles B

Este apartado describe los árboles balanceados, tipo de árboles sobre los que trabaja HEA.

Los árboles B ó árboles balanceados son árboles multicamino con una construcción especial que permite mantenerlos balanceados a bajo costo. [8]

Los árboles están formados por nodos. Un nodo es la unidad sobre la que se construye un árbol y puede tener cero o más nodos conectados a él.
Los nodos de un árbol contienen elementos. Se denomina elemento de un nodo a las claves o índices que, como se menciona en el capítulo 3, son los utilizados para mejorar la performance en la búsqueda de información sobre una Base de Datos. Por lo tanto, los índices son una porción de información que se utiliza para referenciar un registro de una Base de Datos, los cuales se

organizan en estructuras no lineales llamadas árboles, almacenándose en los nodos que componen a dicha estructura.

Resulta importante volver a hacer hincapié en un concepto visto en el apartado anterior que tiene que ver con el orden de un árbol. Una de las características de los árboles multicamino es que cada nodo puede tener cero o más nodos conectados a él llamados nodos hijos. El orden del árbol es el que determina la cantidad de nodos hijos que podrá tener cada nodo. De esta manera, por ejemplo, en un árbol balanceado de orden cinco, cada nodo podrá tener como máximo cinco nodos hijos y cuatro elementos. Dicho de otra manera, el orden del árbol es el que determinará la cantidad de punteros que podrá tener como máximo cada nodo componente del árbol y la cantidad máxima de elementos que podrá contener. La figura 4.4.1 muestra un nodo de un árbol balanceado de orden 5.

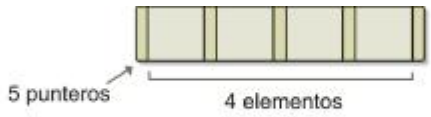

Figura 4.4.1

Un árbol B de orden M posee las siguientes propiedades básicas:

1. Cada nodo del árbol puede contener, a lo sumo, M nodos descendientes y M-1 elementos.
2. La raíz no posee descendientes directos o tiene al menos dos.
3. Un nodo con x descendientes directos contiene x-1 elementos.
4. Todos los nodos terminales se encuentran al mismo nivel.

Analizando las propiedades anteriores, la primera de ellas determina que si el orden de un árbol B es, por ejemplo 256, esto indica que la máxima cantidad de descendientes de cada nodo es 256, porque este valor determina la cantidad de punteros disponibles.

La segunda propiedad posibilita que la raíz no disponga de descendientes, pero cuando sea necesario deberá contar con dos, como mínimo. En ningún caso, la raíz puede tener un solo descendiente.

La tercera característica determina que cualquier nodo que posea una cantidad determinada de hijos, por ejemplo 256, necesariamente debe contener 255 elementos, es decir, uno menos.

La última propiedad establece el balanceo del árbol. La distancia desde la raíz a cada nodo terminal debe ser la misma. Esta característica permite analizar la performance de un árbol B y saber que esta performance será respetada sin importar como se construya, a partir de la llegada de los elementos de datos.

4.4.1 - Operaciones sobre árboles B

A continuación se describen las operaciones que se pueden realizar sobre este tipo de árbol. Estas operaciones son la creación, inserción, la eliminación y la búsqueda de elementos.

Al momento de crear un árbol B, es necesario establecer el orden del mismo. Recordemos que el orden del árbol es el que determinará la cantidad de descendientes de cada nodo y la cantidad de elementos que podrá contener, siendo el orden - 1 la cantidad máxima de elementos de cada nodo.

La operación de inserción de elementos se puede dividir en dos casos. Un primer caso sucede cuando el nodo cuenta con lugar suficiente como para almacenar un nuevo elemento. Este caso es el más sencillo y consiste en la inserción del elemento en el nodo correspondiente, para lo cual en primer lugar se debe localizar el nodo que debería contener al nuevo elemento. Una vez ubicado el nodo se debe insertar el nuevo elemento manteniendo el orden. El orden de los elementos almacenados en cada nodo es fundamental. De esta manera los elementos de cada nodo deberán estar ordenados en forma creciente respetando el concepto visto en árboles binarios, donde además del

orden de los elementos de cada nodo, los elementos menores que no formen parte del nodo se ubicarán en la rama izquierda y los mayores en la rama derecha. La figura 4.4.1.1 representa el orden que deben tener los elementos de un árbol B.

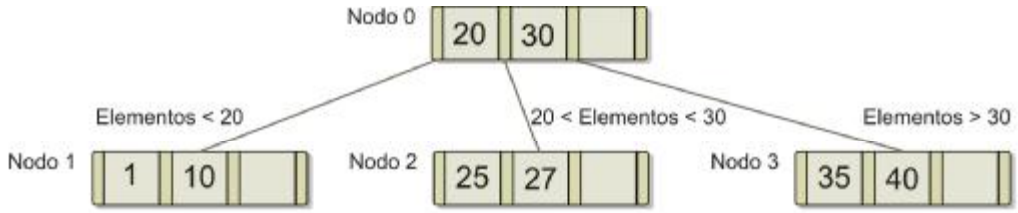

Figura 4.4.1.1

El segundo caso es el más complicado y ocurre cuando se debe insertar un elemento en un nodo que se encuentra completo. Este caso produce overflow y la solución consiste en realizar división y luego promoción.

Ante la ocurrencia de overflow el proceso es el siguiente:

1. Se crea un nodo nuevo
2. La primera mitad de las claves se mantienen en el nodo viejo
3. La segunda mitad de las claves se trasladan al nodo nuevo.
4. La menor de las claves de la segunda mitad se promociona al nodo padre.

El paso 4, denominado promoción, puede provocar un nuevo desborde en el nodo padre. Si esto ocurriese, se repite la operación en el nodo padre desde el paso 1. Se puede notar que este caso tiene una excepción si el nodo en donde se produce overflow es la raíz. En este caso se crean dos nuevos nodos que pasarán a ser hijos de la raíz, distribuyendo los elementos de la primera mitad al nuevo nodo izquierdo, la menor de las claves de la segunda mitad queda en la raíz, y los elementos de la segunda mitad pasarán a ser del segundo nodo

creado, el derecho. Es importante notar que en este caso el árbol aumenta su altura.

El caso de desborde de un nodo permite deducir que la evolución de un árbol B es ascendente, es decir, el árbol va creciendo desde los nodos hojas hacia la raíz. Esto hace que se mantenga la propiedad de balanceo del árbol.

Las figuras 4.4.1.2 y 4.4.1.3 ilustran un ejemplo de overflow en la raíz y en un nodo hoja respectivamente.

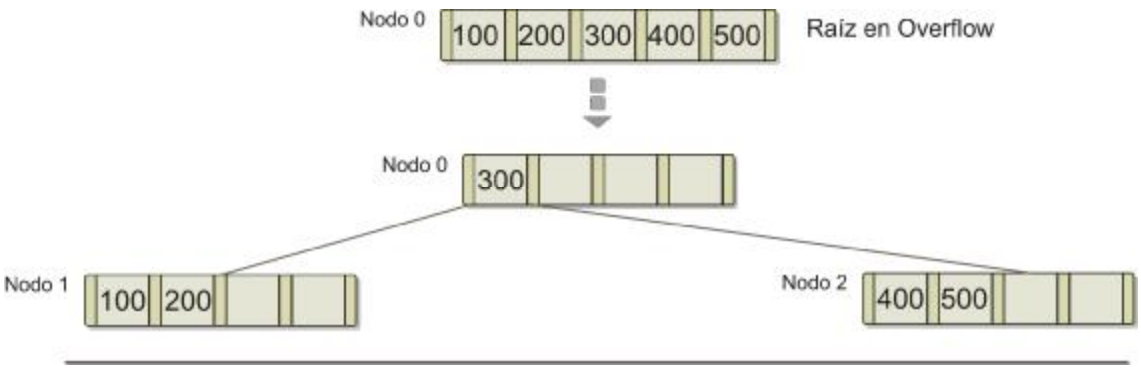

Figura 4.4.1.2

En la figura 4.4.1.2, se puede observar como en un árbol B con orden 5, al insertarse un quinto elemento se produce el desborde de la raíz. Esto deriva en la división del nodo raíz en dos nuevos nodos que serán sus hijos. Es importante poner atención en la distribución de los elementos del árbol. Se puede notar que el elemento que se encontraba en la posición media de la raíz, el 300, queda en ella como elemento separador, mientras que los elementos menores al 300 se insertan en el nuevo nodo del lado izquierdo, el nodo 1, y los elementos mayores son colocados en el nuevo nodo del lado derecho, el nodo 2. De esta manera se mantienen las propiedades que debe cumplir un árbol B.

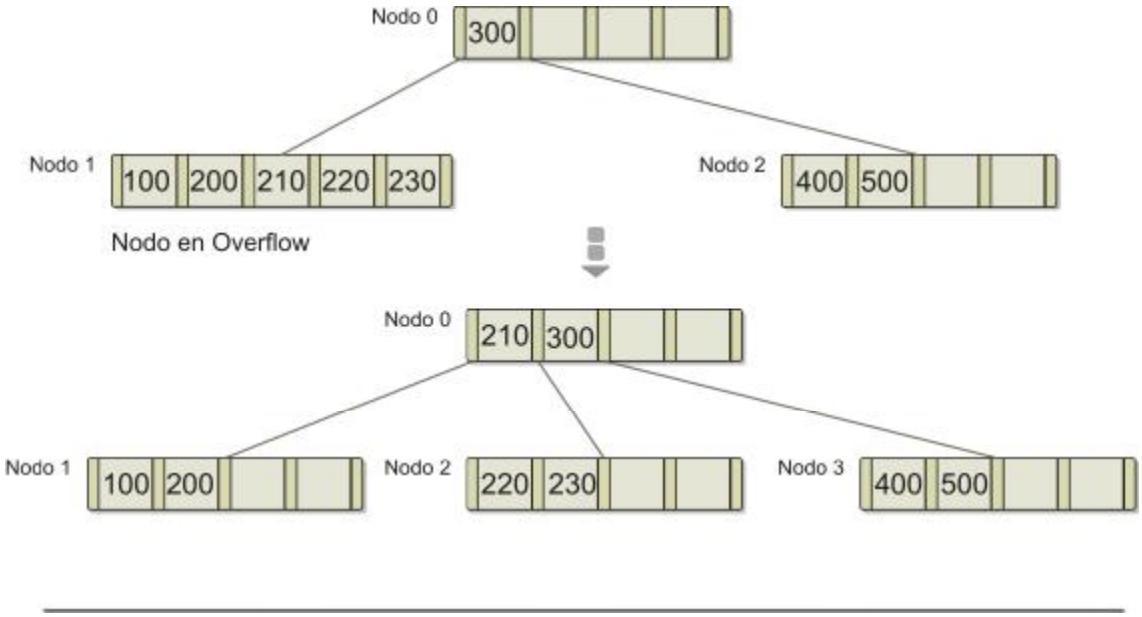

Figura 4.4.1.3

Utilizando también un árbol B de orden 5, la figura 4.4.1.3 muestra el caso en el que el desborde u overflow se produce en un nodo hoja. Este caso deriva en lo que se denomina división y promoción. División porque que el nodo desbordado, el nodo 1, se divide en dos nuevos nodos, y promoción ya que el elemento que se encontraba en la posición media, el 210, se pasa al padre, en este caso la raíz o nodo 0. Luego la distribución del resto de los elementos entre el nodo 1 y el nodo 2 es idéntica al caso de desborde de la raíz, mostrado en la figura 4.4.1.2.

La próxima operación a analizar es la eliminación. Antes de desarrollar esta operación es importante establecer algunas definiciones.

Nodos hermanos: Se denomina nodos hermanos a aquellos nodos que tienen el mismo nodo padre.

Nodos hermanos adyacentes: Se denomina nodos adyacentes hermanos o nodos hermanos adyacentes, a aquellos nodos que siendo hermanos son además dependientes de punteros consecutivos del padre.

Además de las definiciones anteriores es necesario remarcar un concepto que se hereda de los árboles binarios. El elemento que se desea eliminar debe

estar contenido en un nodo terminal. Si esto no ocurre, debe intercambiarse el elemento en cuestión con el menor de sus hijos mayores o rama derecha.

Para una mejor comprensión de la operación de eliminación, es conveniente separarla en los distintos casos posibles.

Un primer caso, el más sencillo, se produce cuando se desea eliminar un elemento, y no provoca underflow en el nodo que lo contiene. Un nodo entra en underflow si al producirse una baja de un elemento en él contenido, la cantidad total de elementos de ese nodo, queda por debajo del mínimo permitido, según lo describen las propiedades de los árboles B enunciadas al principio de este apartado. En este caso, la operación de eliminación consiste solamente en quitar el elemento del nodo, y reacomodar los elementos restantes para mantener el orden, en caso de que sea necesario. La figura 4.4.1.4 muestra un ejemplo de eliminación para este caso.

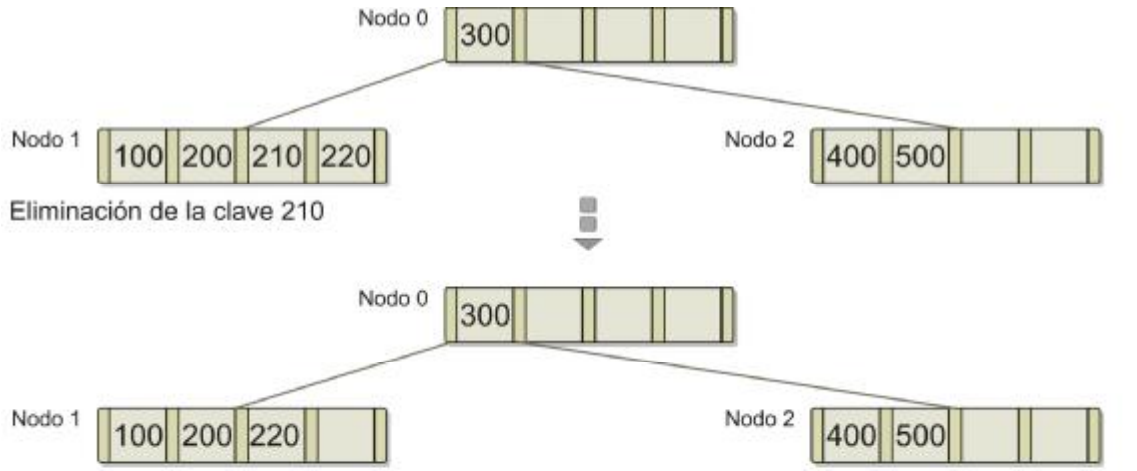

Figura 4.4.1.4

El segundo caso en la operación de baja ocurre al eliminar un elemento y se produce underflow en el nodo que lo contiene. Nuevamente resulta adecuado dividir este caso en dos subcasos más. El primer subcaso se produce si alguno de los hermanos adyacentes del nodo en underflow puede cederle un elemento. Un hermano adyacente podrá ceder un elemento si al hacerlo no

entra en underflow. En este caso se produce lo que se denomina redistribución. El elemento más grande del hermano adyacente izquierdo o el más chico del hermano adyacente derecho (según que hermano adyacente se elija para redistribuir) pasa al padre y el elemento divisor del padre pasa al nodo en underflow. De esta manera se mantienen las propiedades de un árbol B. Se puede observar la figura 4.4.1.5 que ilustra un caso de redistribución.

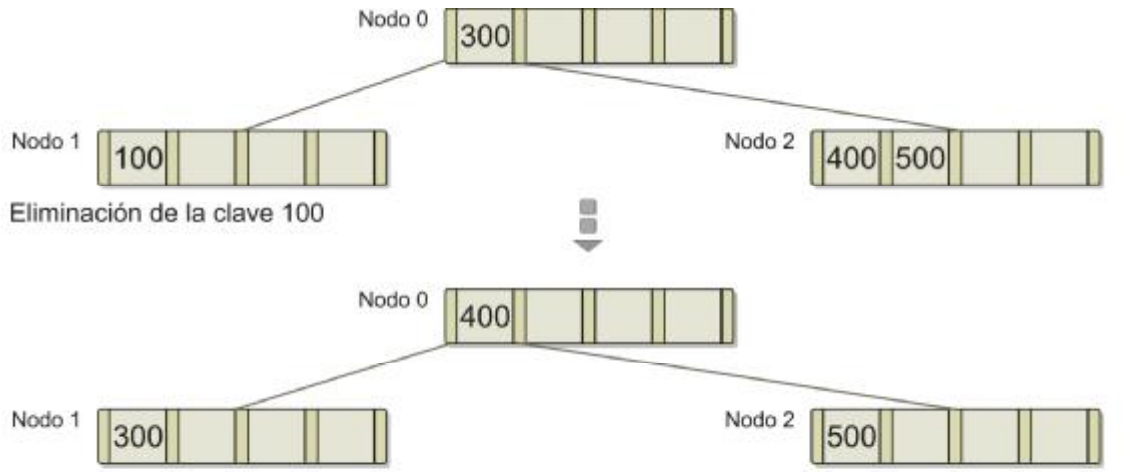

Figura 4.4.1.5

Como se puede observar en la figura 4.4.1.5, al eliminarse la clave 100, el nodo que la contiene entra en underflow. Como el hermano adyacente derecho, el nodo 2, puede ceder un elemento sin quedar por debajo del mínimo de elementos permitidos, pasa la clave 400 al padre, en este caso la raíz, y el elemento 300 de la raíz al nodo en underflow, manteniendo de esta manera, las propiedades de un árbol B.

El segundo subcaso ocurre cuando ningún hermano adyacente del nodo en underflow puede ceder un elemento sin quedar por debajo del mínimo permitido. En este caso se produce lo que se denomina fusión. La fusión actúa de la siguiente manera: se unen dos nodos hermanos adyacentes más el elemento divisor del padre y forman un solo nodo con la unión de todos los elementos involucrados. La figura 4.4.1.6 muestra un caso de fusión en un árbol B.

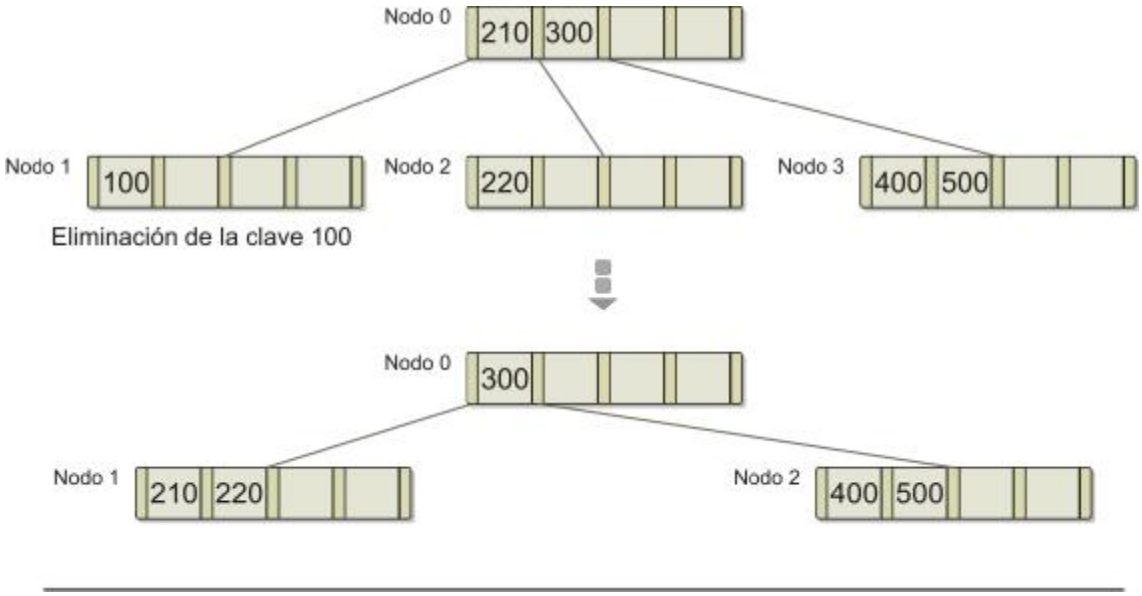

Figura 4.4.1.6

Como se muestra en la figura 4.4.1.6, al eliminar el elemento 100, el nodo que lo contiene queda por debajo del mínimo permitido. Como el hermano adyacente derecho, el nodo 2, no puede ceder un elemento, ya que si lo hiciera también entraría en underflow, se fusionan ambos nodos, uniendo en el nodo resultante los elementos de ambos, en este caso solo el 220, más el elemento separador que se encontraba en el padre, la clave 210.

La figura 4.4.1.7 describe el caso de una fusión en donde la raíz posee solo dos hijos. Se puede observar que al producirse la fusión de los mismos, el árbol disminuye su altura.

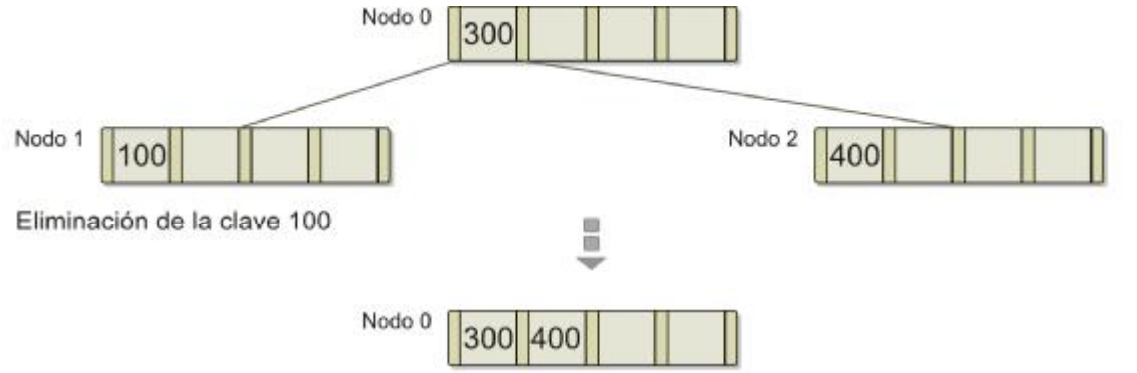

Figura 4.4.1.7

La siguiente y última operación posible sobre un árbol B es la búsqueda. Para localizar un elemento contenido en un árbol B se procede de la siguiente manera: se busca el elemento en cuestión desde el nodo raíz. En caso de encontrarlo en dicho nodo, se retorna una condición de éxito. Si no se encuentra, se procede a buscar en el nodo inmediato siguiente que debería contener el elemento, procediendo de esa forma hasta encontrar el elemento, o hasta encontrar un nodo sin hijos que no incluya al elemento. Para determinar el camino de búsqueda, se aprovecha el orden en que están almacenados los elementos, de tal manera, se comienza desde la raíz, se analizan los elementos de la misma y si el buscado es menor a dichos elementos se repite la búsqueda en la rama izquierda y si es mayor en la rama derecha. La figura 4.4.1.8 describe el camino en la búsqueda del elemento 500 en un árbol B.

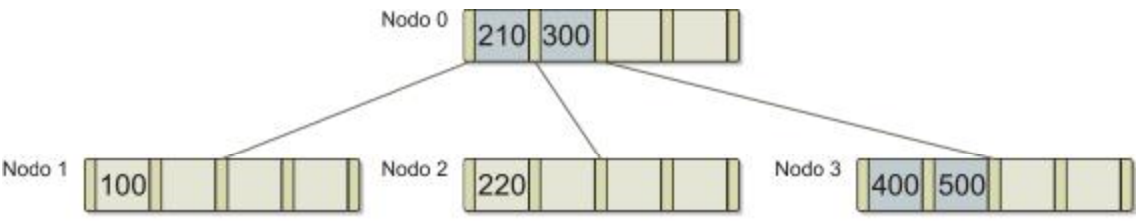

Figura 4.4.1.8

4.4.2 - Árboles B*

Los árboles B* presentan una variante sobre los árboles B. La consideración efectuada para avanzar sobre una estructura B* parte del análisis realizado en el proceso de baja de información. Resumiendo este caso, el tratamiento de una saturación u overflow en un árbol B, genera una sola operación, la división. El tratamiento de una insuficiencia genera dos operaciones, redistribución y/o concatenación (o fusión). Claramente se puede observar que división y concatenación son operaciones opuestas. Entonces, el proceso de tratamiento de saturaciones en un árbol B no plantea una operación equivalente a la redistribución en el proceso de baja.

De esta manera, antes de dividir y generar nuevos nodos se dispone de una variante, redistribuir también ante una saturación. Esta acción demorará la generación de nuevos nodos y por ende tendrá el efecto de aumentar en forma más lenta la cantidad de niveles del árbol. Si la cantidad de niveles del árbol crece más lentamente, la performance final de la estructura para recuperación de información es mejor.

Si se aplica el concepto de redistribuir, cuando un nodo se completa, reubica sus elementos utilizando un nodo adyacente hermano. Cuando no sea posible esta redistribución, se estará ante una situación donde tanto el nodo que genera overflow como su adyacente hermano están completos. Esto abre la posibilidad de dividir partiendo de dos nodos completos y generando tres nodos completos en dos tercera partes (2/3).

Un árbol B* es un árbol balanceado con las siguientes propiedades especiales:

- ✓ Cada nodo del árbol puede contener como máximo M descendientes y M-1 elementos.
- ✓ La raíz no posee descendientes o tiene al menos dos.
- ✓ Un nodo con x descendientes contiene x-1 elementos.
- ✓ Todos los nodos terminales se encuentran al mismo nivel.

La operación de búsqueda sobre un árbol B* es similar a la presentada anteriormente para árboles B. La naturaleza de ambos árboles para localizar un elemento no presenta diferencias de tratamiento. La búsqueda comienza en el nodo raíz y se avanza hasta encontrar el elemento, o hasta llegar a un nodo terminal y no poder continuar con el proceso.

La operación de baja resulta, nuevamente similar, tanto para los casos donde no se genera underflow como para aquellos donde sí se genera insuficiencia. En este último caso, la primera opción consiste en redistribuir, o en su defecto se deberá fusionar, basados en los principios definidos para árboles B.

En lo que sigue de este apartado haremos hincapié en la operación de inserción que es la que presenta diferencias con respecto a los árboles B. En los árboles B* surge un nuevo concepto que se denomina política. La política de un árbol B* determina con cuál de los hermanos adyacentes se realizará la redistribución en el caso de producirse overflow. De esta manera las posibles políticas de redistribución de un árbol B* son las siguientes:

- ✓ Política Derecha: como primera acción se intenta redistribuir con el hermano adyacente derecho, llevándose a cabo dicha acción en caso de ser posible. Si la redistribución con el hermano adyacente derecho no puede llevarse a cabo, se realiza una fusión con el hermano adyacente derecho, haciendo de dos nodos, tres, dos tercios llenos.

- ✓ Política izquierda: como primera acción se intenta redistribuir con el hermano adyacente izquierdo, llevándose a cabo dicha acción en caso de ser posible. En caso de no poderse, se realiza una fusión con el hermano adyacente izquierdo, haciendo de dos nodos, tres, dos tercios llenos.

- ✓ Política Izquierda o Derecha: determina que en primera instancia la redistribución debe realizarse con el hermano adyacente izquierdo, en caso de no ser posible, con el hermano adyacente derecho y en caso de

no poder redistribuir con ninguno, se fusiona con el hermano adyacente izquierdo, haciendo de dos nodos, tres, dos tercios llenos.

- ✓ Política Derecha o Izquierda: determina que la redistribución debe realizarse primero con el hermano adyacente derecho, en caso de no ser posible, con el hermano adyacente izquierdo y en caso de no poder redistribuir con ninguno, se fusiona con el hermano adyacente derecho, haciendo de dos nodos, tres, dos tercios llenos.
- ✓ Política Izquierda y Derecha: como primera acción se intenta redistribuir con el hermano adyacente izquierdo si no es posible se intenta redistribuir con el hermano adyacente derecho y en caso de no poder redistribuir con ninguno, se fusionan los tres nodos, haciendo cuatro, tres cuatros llenos

Por lo tanto al insertarse un elemento en un árbol B *, la primera acción que se toma es buscar el nodo en donde corresponde insertar el elemento, al igual que los árboles B. Una vez encontrado, si la inserción del nuevo elemento en el nodo, produce overflow, es cuando entra en consideración la política seleccionada para el árbol. De esta manera se actúa según lo explicado en cada una de las políticas. Por otro lado si el nuevo elemento no produjese desborde en el nodo, se inserta normalmente reordenando los elementos almacenados en el nodo si fuese necesario.

A continuación se presentan una serie de figuras que ilustran el comportamiento de los árboles B* en la operación de inserción.

La figura 4.4.2.1 muestra un ejemplo de inserción con desborde de un nodo en un árbol B * con política izquierda.

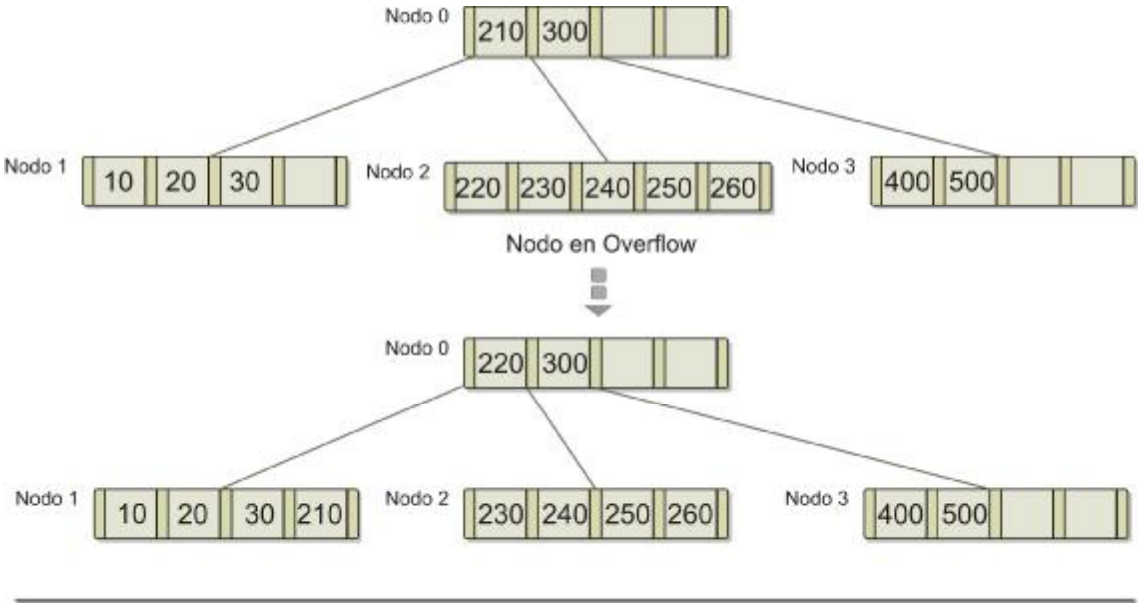

Figura 4.4.2.1

Como se puede observar en la figura 4.4.2.1 al tratarse de un árbol B* con política izquierda y orden 5, cuando se inserta el elemento 260, el nodo 2 entra en overflow. Como el hermano izquierdo del nodo desbordado puede aceptar un elemento sin entrar en overflow, se produce una redistribución, pasando el elemento más chico del nodo 2, el 220, a la raíz y el elemento separador de la raíz, el 210, al nodo 1, manteniendo de esta manera las propiedades que debe cumplir un árbol B*. Es importante notar que la redistribución es posible porque el hermano izquierdo del nodo en overflow permite la incorporación de un elemento sin desbordarse.

Manteniendo las características del árbol de la figura 4.4.2.1, la imagen 4.4.2.2 muestra el caso en el que el hermano izquierdo del nodo en overflow no puede aceptar un nuevo elemento produciéndose lo que se denomina fusión, creando a partir de dos nodos llenos, tres nodos dos tercios llenos.

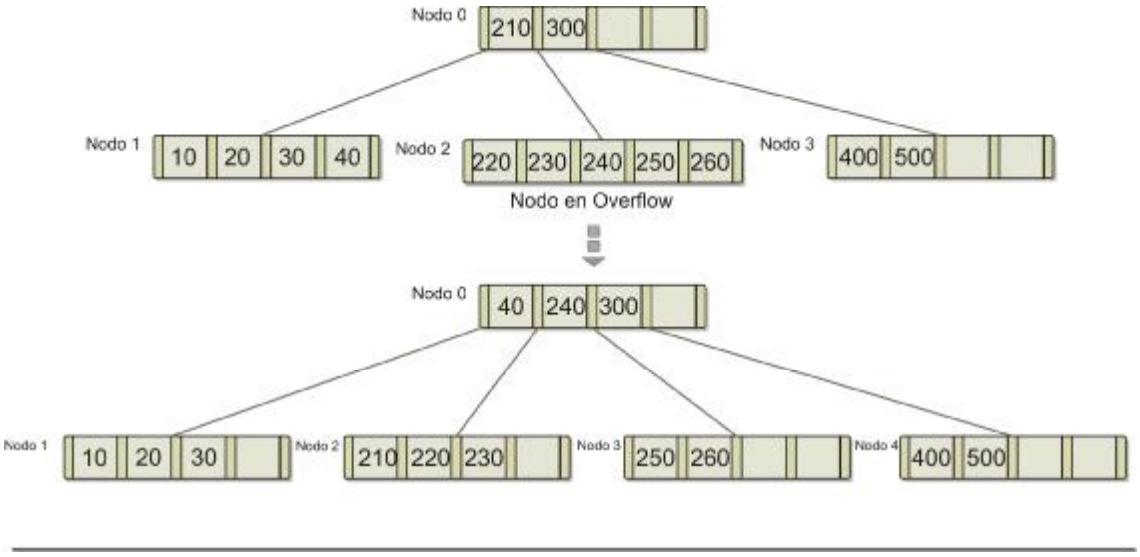

Figura 4.4.2.2

En la imagen 4.4.2.3 se presenta el caso de un árbol B* con orden 5 y con política izquierda o derecha, en donde el nodo 2 entra en overflow. En primer lugar se determina si el hermano izquierdo del nodo desbordado, el nodo 1, acepta un elemento sin entrar en overflow. Como esto no es posible, se observa si el hermano derecho, el nodo 3, permite la inserción de un nuevo elemento sin desbordar. Debido a que el nodo 3 tiene lugar suficiente como para aceptar un nuevo elemento, se produce una redistribución a derecha, pasando el elemento más grande del nodo en overflow, el 260, al nodo padre, en este caso el nodo 0, y el elemento divisor del padre, el 300, al nodo 3, manteniéndose así las propiedades que debe cumplir un árbol B*.

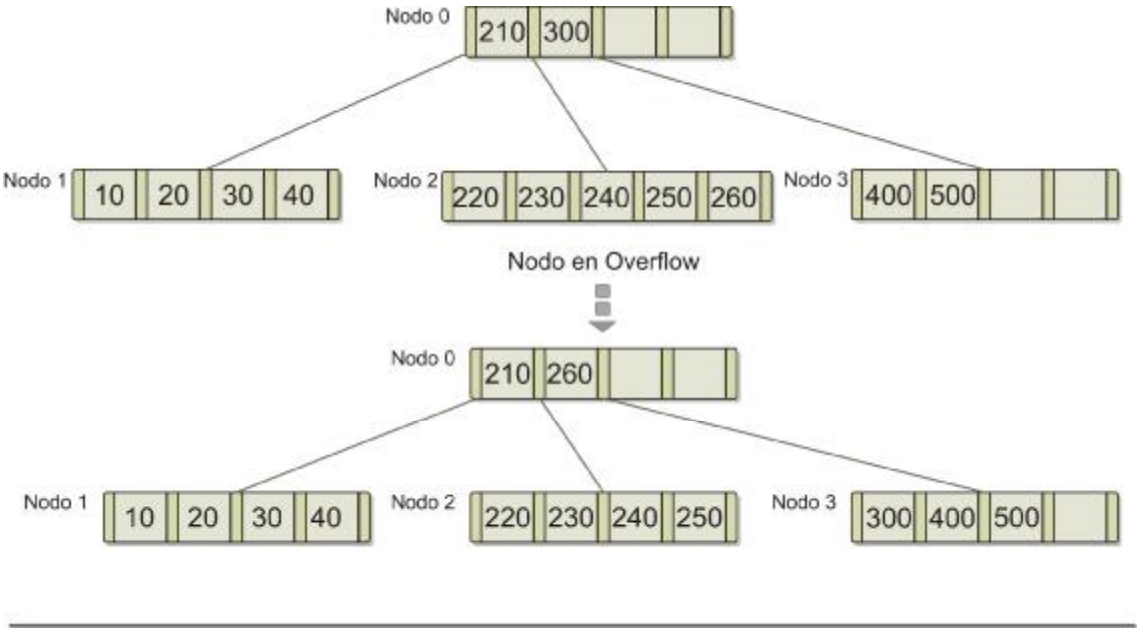

Figura 4.4.2.3

Siguiendo el ejemplo anterior, otro de los casos posibles, es que al determinar como segunda alternativa si el hermano derecho permite la inserción de un nuevo elemento sin entrar en overflow, esto no sea posible. Debido a que ni el hermano izquierdo ni el derecho del nodo desbordado permiten la inserción de un nuevo elemento sin desbordar se produce una fusión con el hermano izquierdo. La figura 4.4.2.4 muestra un ejemplo de este caso.

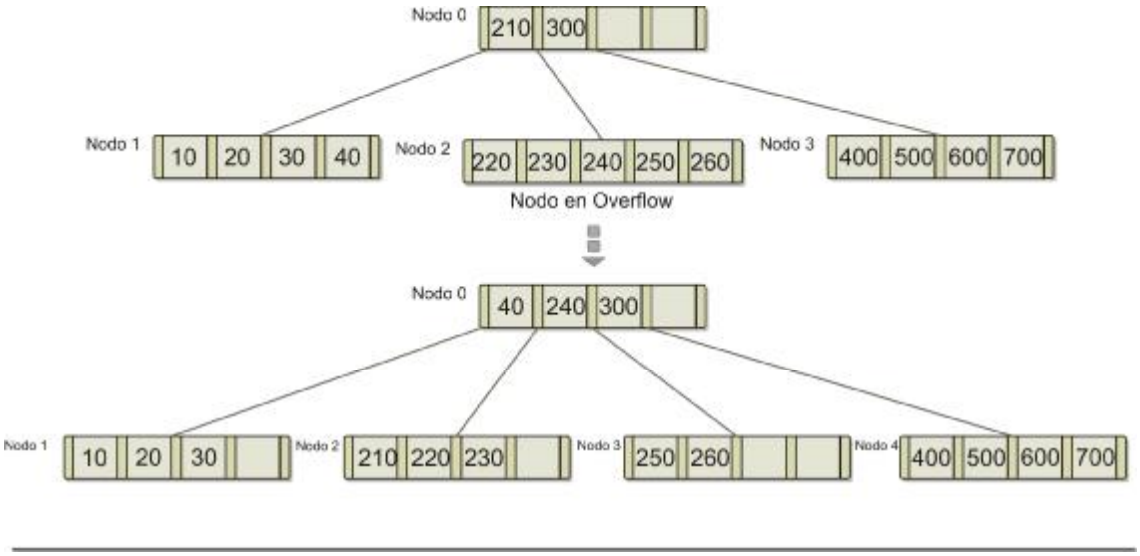

Figura 4.4.2.4

La última política a tratar es la política izquierda y derecha. Para ilustrar este caso la figura 4.4.2.5 muestra un árbol B* de orden 4 en donde el nodo 2 entra en overflow. Como puede observarse, tanto el nodo 1 como el nodo 3, hermanos adyacentes del nodo en sobre flujo, no aceptan un nuevo elemento, por lo tanto, es imposible realizar una redistribución. Este es el caso en el que se produce la fusión de los tres nodos anteriormente mencionados, formando cuatro nodos.

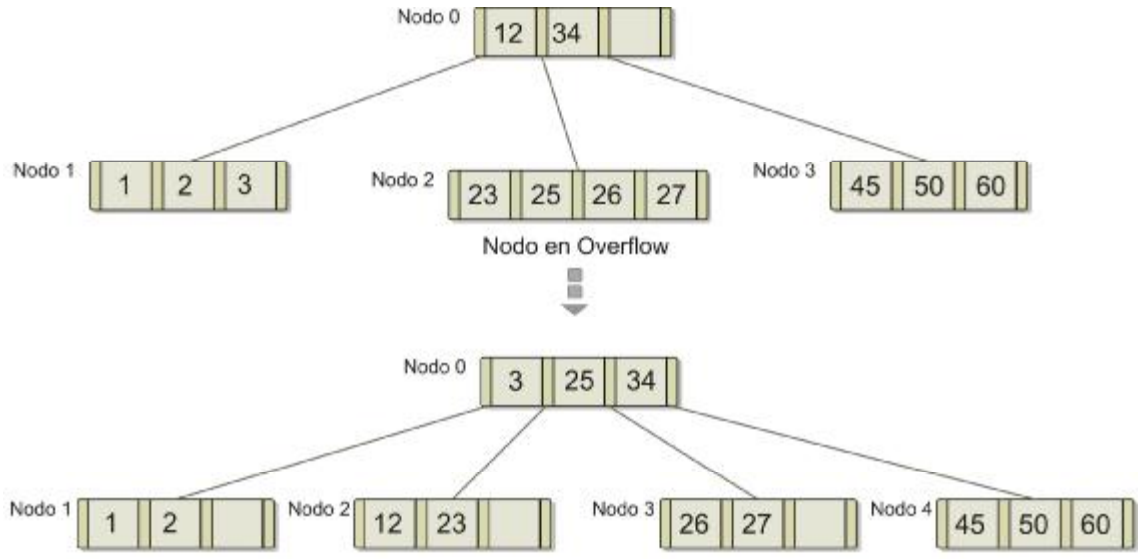

Figura 4.4.2.5

4.4.3 - Árboles B +

El objetivo principal de los árboles B+ es poder hacer, a bajo costo, un acceso secuencial ordenado a todos los elementos.

Un árbol B+ es un árbol multicamino con las siguientes propiedades:

- ✓ Cada nodo del árbol puede contener como máximo M descendientes y M-1 elementos.
- ✓ La raíz no posee descendientes o tiene al menos dos.
- ✓ Un nodo con x descendientes contiene x-1 elementos.
- ✓ Todos los nodos terminales se encuentran al mismo nivel.
- ✓ Los nodos terminales representan un conjunto de datos y son enlazados entre ellos.

Si se compara la definición anterior con la resultante para árboles B encontrará similitudes, excepto en la última propiedad.

Es esta última propiedad la que establece la principal diferencia entre un árbol B y un árbol B+. Para poder realizar acceso secuencial ordenado a todos los registros del archivo, es necesario que cada elemento aparezca almacenado en un nodo terminal. Así, los árboles B+ diferencian los elementos que constituyen datos de aquellos que son separadores.

Al comenzar la creación de un árbol B+, el único nodo disponible, el nodo raíz, actúa tanto como punto de partida para búsquedas como para acceso secuencial, tal como se muestra en la figura 4.4.3.1.

Figura 4.4.3.1

En el momento que el nodo se satura, se produce una división. Se genera un nuevo nodo terminal donde se redistribuyen los elementos.

Como lo muestra la figura 4.4.3.2, ahora se dispone de tres nodos, los nodos terminales contendrán todos los elementos, y el nodo raíz tiene al elemento que actúa como separador. Se debe notar que la clave utilizada como separador es la misma que está contenida en el nodo terminal, es decir, se utiliza una copia del elemento y no el elemento en sí.

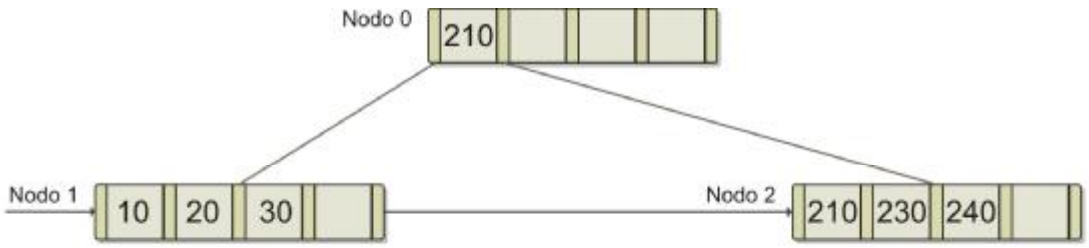

Figura 4.4.3.2

Es importante notar en la figura 4.4.3.2 el enlace que existe entre los nodos del último nivel. Esto representa la posibilidad de acceso secuencial a los datos del árbol comenzando la lista en el nodo hoja que se encuentra más a la izquierda, en este caso el nodo 1.

El proceso de creación del árbol B+ sigue los lineamientos presentados anteriormente para árboles B.

Los elementos siempre se insertan en nodos terminales. Si se produce una saturación, el nodo se divide y se promociona una copia (aquí yace la diferencia) del menor de los elementos mayores, hacia el nodo padre. Si el padre no tuviera espacio para contenerlo se dividirá nuevamente.

Se debe notar que en caso de dividir un nodo no terminal, se debe promocionar al padre el elemento en sí y no una copia del mismo, es decir, sólo ante la división de un nodo terminal se debe promocionar una copia.

Para borrar un elemento de un árbol B+ siempre se borra de un nodo terminal, y si hubiese una copia de ese elemento en un nodo no terminal, esta copia se mantiene porque sigue actuando como separador. Este caso puede observarse en la figura 4.4.3.3.

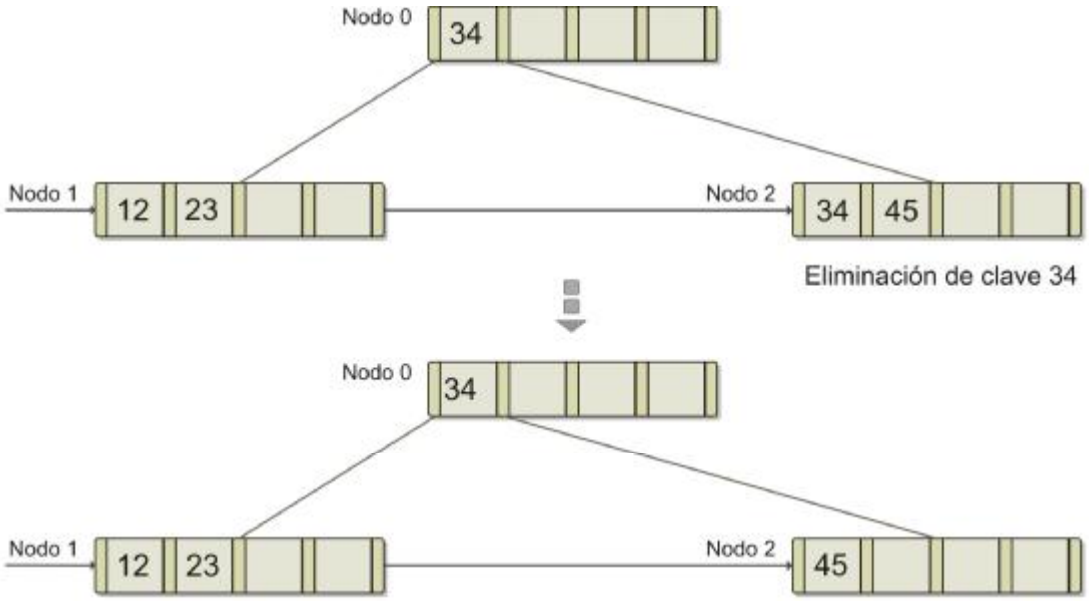

Figura 4.4.3.3

La figura 4.4.3.3 muestra un árbol B + con orden 5 en donde se elimina la clave 34 contenida en el nodo 2. Como puede observarse al borrar la clave 34, el nodo 2 no entra en underflow, razón por la cual el elemento puede borrarse normalmente. Por otro lado es importante notar que al producirse la eliminación de la clave 34 no se borra la copia contenida en el nodo 0, la cual continúa siendo separador del árbol.

Capítulo 5 – HEA (Herramienta de software para la Enseñanza de Árboles B).

La realización de una herramienta de software que sirva como complemento académico, plantea un desafío en lo que respecta tanto a la parte visual como a la parte funcional. El objetivo de este capítulo es detallar aspectos visuales y funcionales de HEA, de manera tal de expresar en profundidad las alternativas que brinda la herramienta como complemento educativo.

En una primera instancia se presentan las diferentes alternativas funcionales que la herramienta brinda al usuario. Luego se describen las decisiones tomadas respecto al diseño gráfico de la herramienta.

5.1 Descripción Funcional

La figura 5.1.1 muestra una presentación de la interfaz propuesta en HEA. Para facilitar la identificación de las tareas, se han agregado a la imagen algunas referencias numéricas.

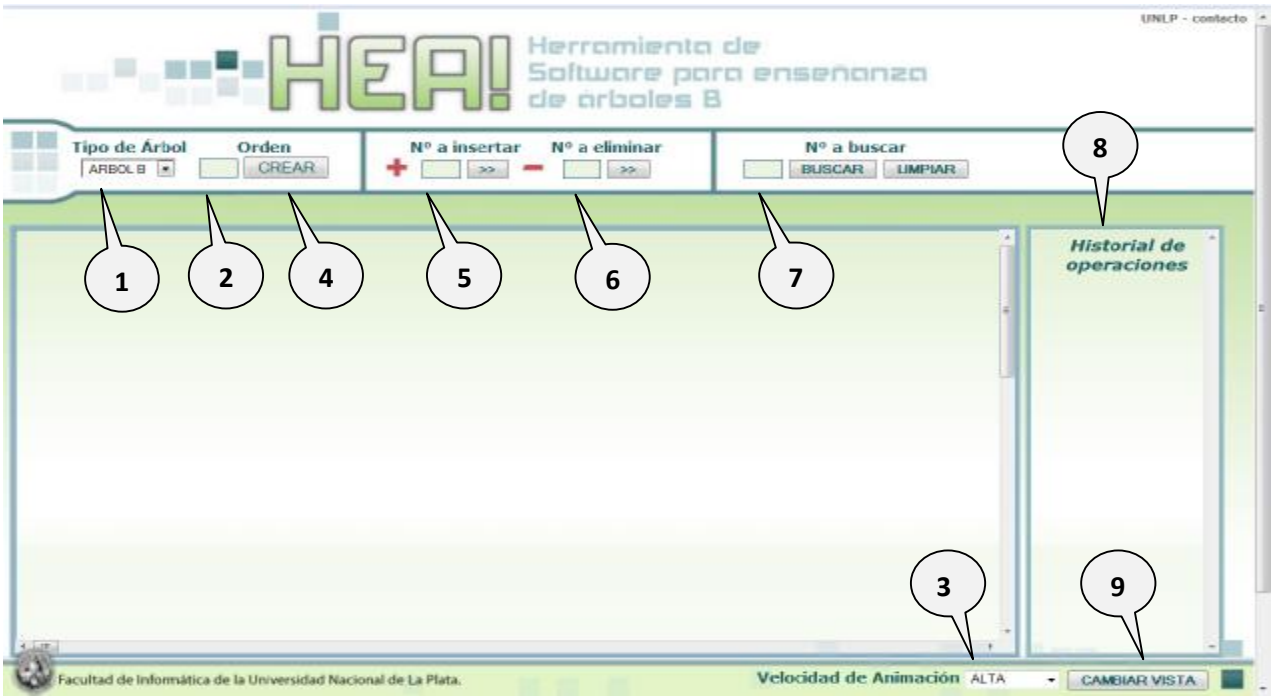

Figura 5.1.1

Las operaciones de HEA pueden agruparse de acuerdo a dos criterios: operaciones de configuración de parámetros, conforman el primer grupo, y operaciones básicas sobre el árbol el segundo.

5.1.1 - Grupo de configuración de parámetros

Un árbol balanceado, como se ha descripto oportunamente, puede ser de tipo B, B+ o B*. Una característica común de los diferentes tipos de árboles es el orden, que como bien se mencionara en los capítulos anteriores, determina la cantidad de descendientes que puede tener un nodo y la cantidad máxima de elementos que cada nodo puede contener. Estas características pueden ser configuradas en HEA a lo largo de una sesión de aprendizaje, mediante el ajuste de ciertos parámetros.

Dentro del grupo de configuración de parámetros se encuentran aquellas operaciones que determinan las características que tendrá un árbol a lo largo de una sesión de aprendizaje. Se denomina sesión de aprendizaje a todas las operaciones realizadas con un mismo conjunto de parámetros.

La primera operación de configuración es la selección del tipo de árbol (referencia 1 figura 5.1.1) en la cual el usuario debe indicar si desea trabajar sobre un árbol B, B+ o B*. Una particularidad de los árboles B* es que necesitan un parámetro adicional ya que deben considerarse ciertas políticas para su funcionamiento, según lo explicado en el capítulo 4. Por lo tanto, si el usuario selecciona este tipo de árbol, se despliega un conjunto de políticas tal como se muestra en la figura 5.1.2. Las políticas de un árbol B* también forman parte del grupo de configuración de parámetros.

Figura 5.1.2

Una vez que el usuario ha seleccionado el tipo de árbol con el que desea trabajar en la sesión de aprendizaje, es necesario configurar el orden del mismo (referencia 2 figura 5.1.1). La herramienta permite que el usuario seleccione un orden dentro de un rango comprendido entre 4 y 9. Si bien en una implementación real de Base de Datos este valor será extremadamente mayor, por tratarse de una herramienta netamente educativa se decidió que el orden se encuentre dentro de valores que permitan al usuario un seguimiento visual sencillo, sin perjuicio de entender el funcionamiento de las estructuras del tipo árbol.

Para finalizar la etapa de configuración, el usuario selecciona la velocidad de animación (referencia 3 figura 5.1.1). Este parámetro permite adaptar la herramienta de acuerdo al nivel de conocimiento que posea el alumno modificando la velocidad con que se realizan las animaciones explicativas. Una

animación lenta permitirá afianzar los conocimientos, mientras que una animación rápida, evitará que un alumno avanzado pierda tiempo con la evolución de las animaciones que detallan conocimientos ya adquiridos. La velocidad de animación por defecto que trae la herramienta es *alta,* pudiendo cambiarse a *media* o *baja*. A diferencia del resto de los parámetros pertenecientes al grupo de configuración, la velocidad de animación, podrá cambiarse a lo largo de una sesión de aprendizaje.

Finalizada la configuración, están dadas las condiciones para crear un árbol. Para ello se cuenta con la opción Crear (referencia 4 figura 5.1.1). Cuando se selecciona esta opción se presentará una animación de la raíz del árbol respetando el orden seleccionado previamente como se muestra en la figura 5.1.3. En caso de que alguno de los parámetros de configuración no sea elegido correctamente, HEA mostrará mensajes en pantalla que le indicarán al usuario tal situación. La figura 5.1.3 muestra la creación de un árbol del tipo B con un orden 4.

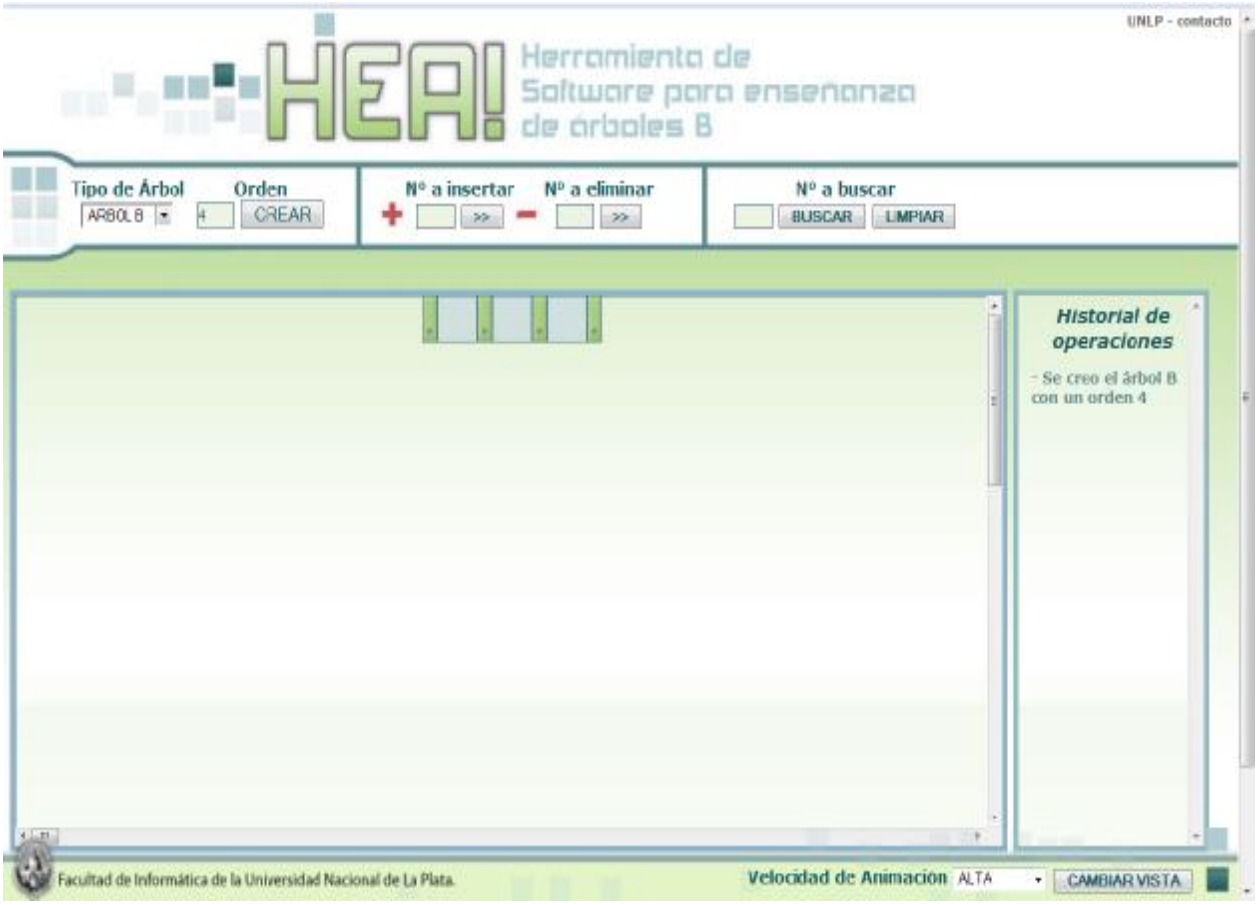

Figura 5.1.3

5.1.2 - Grupo de operaciones básicas

El grupo de operaciones básicas está formado por aquellas que hacen a la evolución de la estructura, como ser la inserción, eliminación y búsqueda. Si bien estas operaciones pertenecen a un grupo particular, son dependientes del grupo de configuración visto en el apartado anterior. Por lo tanto las operaciones básicas tendrán diferente comportamiento en función del tipo de árbol y del orden que se haya configurado en la primera etapa.

No es objeto de este apartado describir el comportamiento de un árbol, tema perteneciente al capítulo 4. Sin embargo, se hará un repaso de las operaciones básicas con el objetivo de comprender como utilizarlas y cuáles fueron las decisiones tomadas en el diseño de la herramienta.

La primera operación del grupo consiste en la inserción de elementos (referencia 5 figura 5.1.1). HEA permite la inserción de claves numéricas con la finalidad de que el alumno comprenda fácilmente el ordenamiento de las mismas dentro de un árbol. A medida que se ingresan claves, HEA muestra la evolución de la estructura a través de animaciones.

Una de las decisiones tomadas es en relación al desborde de un nodo. Como se viera en al capítulo 4, frente a esta situación y al tratarse de árboles B o B+, se produce lo que se denomina división y promoción. Al momento de distribuir los índices en los nodos creados, si la cantidad de claves es par, se sobrecargará el nodo izquierdo. Sin embargo podría haberse tomado como alternativa sobrecargar el nodo derecho lo cual no influye en el correcto funcionamiento del árbol como tampoco en el aprendizaje del tema. Para aclarar dicha situación se puede observar la figura 5.1.4 en la cual se muestra un árbol B de orden 4 donde la raíz ha sufrido una división en dos nuevos nodos. Debe observarse que al momento de distribuir los elementos en los dos nuevos nodos creados, el nodo 0, contiene dos claves, mientras que el nodo 1, contiene solo una.

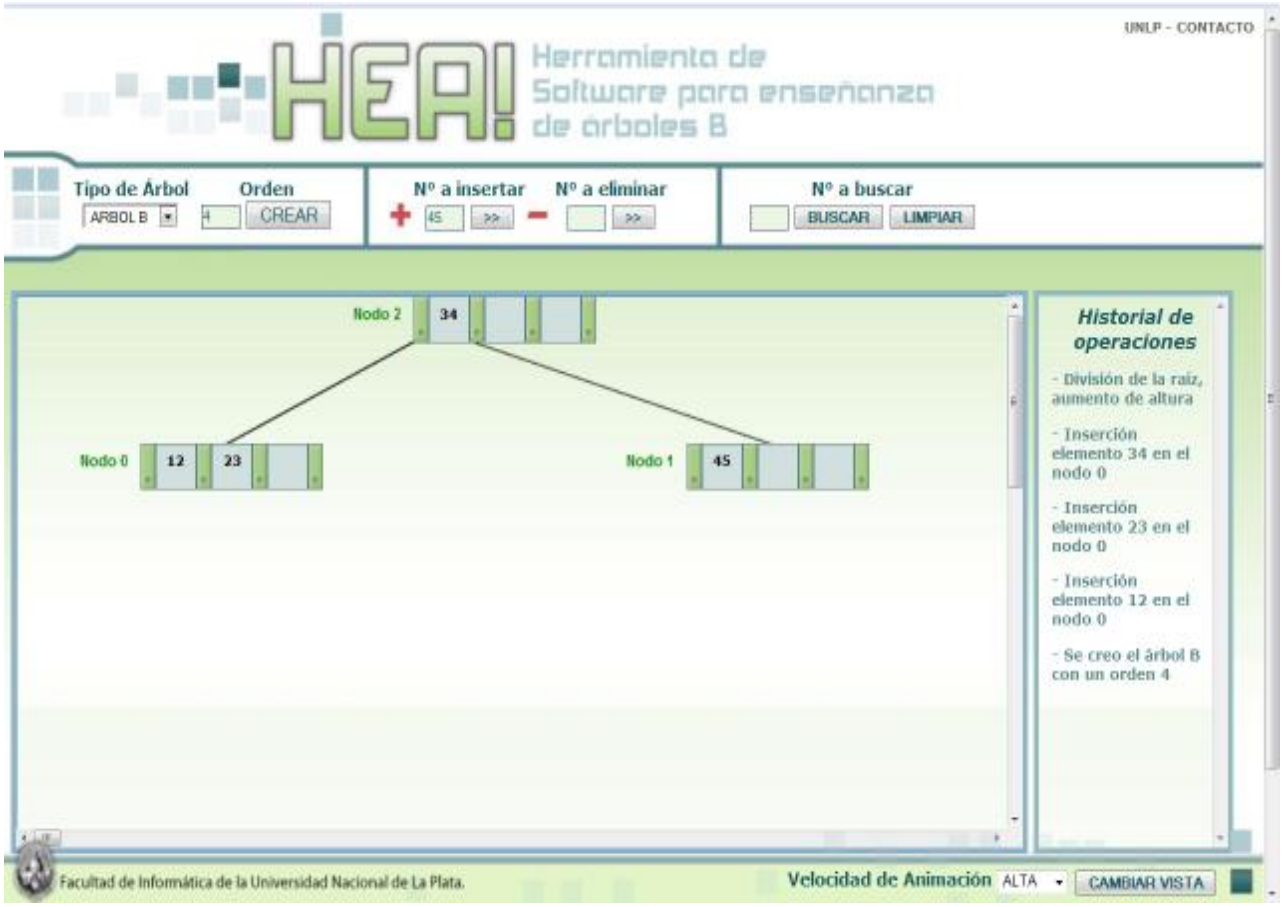

Figura 5.1.4

Siguiendo con las operaciones posibles se describe la operación de eliminación de claves. La eliminación tiene sentido sólo cuando el árbol contiene al menos una clave (referencia 6 figura 5.1.1). Si el usuario de HEA desea borrar un elemento en un árbol vacio se mostrará un mensaje en pantalla indicando tal situación.

Al borrar una clave, el usuario de HEA, podrá observar el reordenamiento de los elementos del árbol y animaciones que demuestran la fusión de nodos en caso de producirse. Una decisión tomada con respecto a la eliminación radica en la cantidad mínima de elementos o claves que puede contener un nodo. Se ha tomado como convención que dicha cantidad se calcule en función del orden seleccionado por el alumno, siendo:

m = ((o / 2) – 1).
m=cantidad mínima de elementos de un nodo.
o=orden del árbol.

Es importante aclarar esta convención ya que algunos textos mencionan que la cantidad mínima de elementos difiere si se trata de un nodo terminal o de un nodo intermedio.

En el diseño de HEA la cantidad mínima de claves que puede contener un nodo no encuentra diferencias entre nodos intermedios y nodos hojas. Esta decisión no afecta al correcto funcionamiento de los árboles ni tampoco el proceso de aprendizaje y ha sido tomada con la finalidad de facilitar la incorporación de conocimientos, estandarizando para todos los nodos la misma cantidad mínima de elementos permitidos.

Como se ha visto en los capítulos anteriores, una de las principales razones de la utilización de árboles en el manejo de Bases de Datos, es la eficiencia lograda con su uso como estructura de datos para almacenar índices, utilizados en el proceso de búsqueda de información.

La tercera operación del grupo de operaciones básicas tiene que ver con este principio. El usuario de HEA cuenta con la posibilidad de buscar una clave (referencia 7 figura 5.1.1) almacenada en el árbol, observando, a través de una animación, el camino de búsqueda hacia el índice y diferenciando por medio de diferentes colores si la misma ha resultado exitosa o no. En las figuras 5.1.5 y 5.1.6 se representan ambas situaciones.

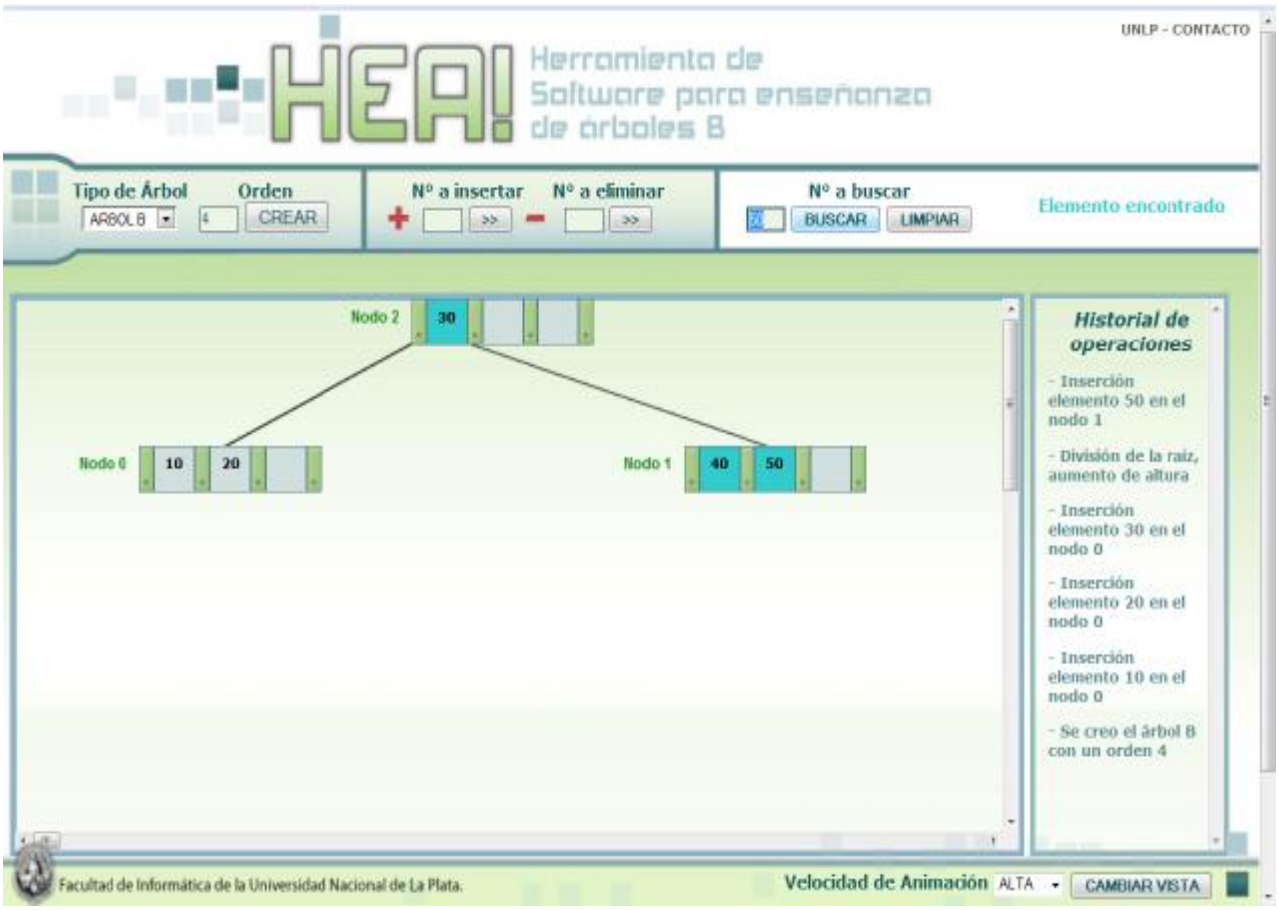

Figura 5.1.5

La figura 5.1.5 muestra el caso de la búsqueda del elemento 50 en un árbol B de orden 4. Debido a que la clave 50 se encuentra en el árbol, en este caso en el nodo 1, HEA muestra el camino de búsqueda hacia dicho índice coloreando la ruta de color azul el cual denota éxito. Además se puede observar la leyenda *Elemento encontrado* en el costado izquierdo de la zona de búsqueda en la paleta de operaciones.

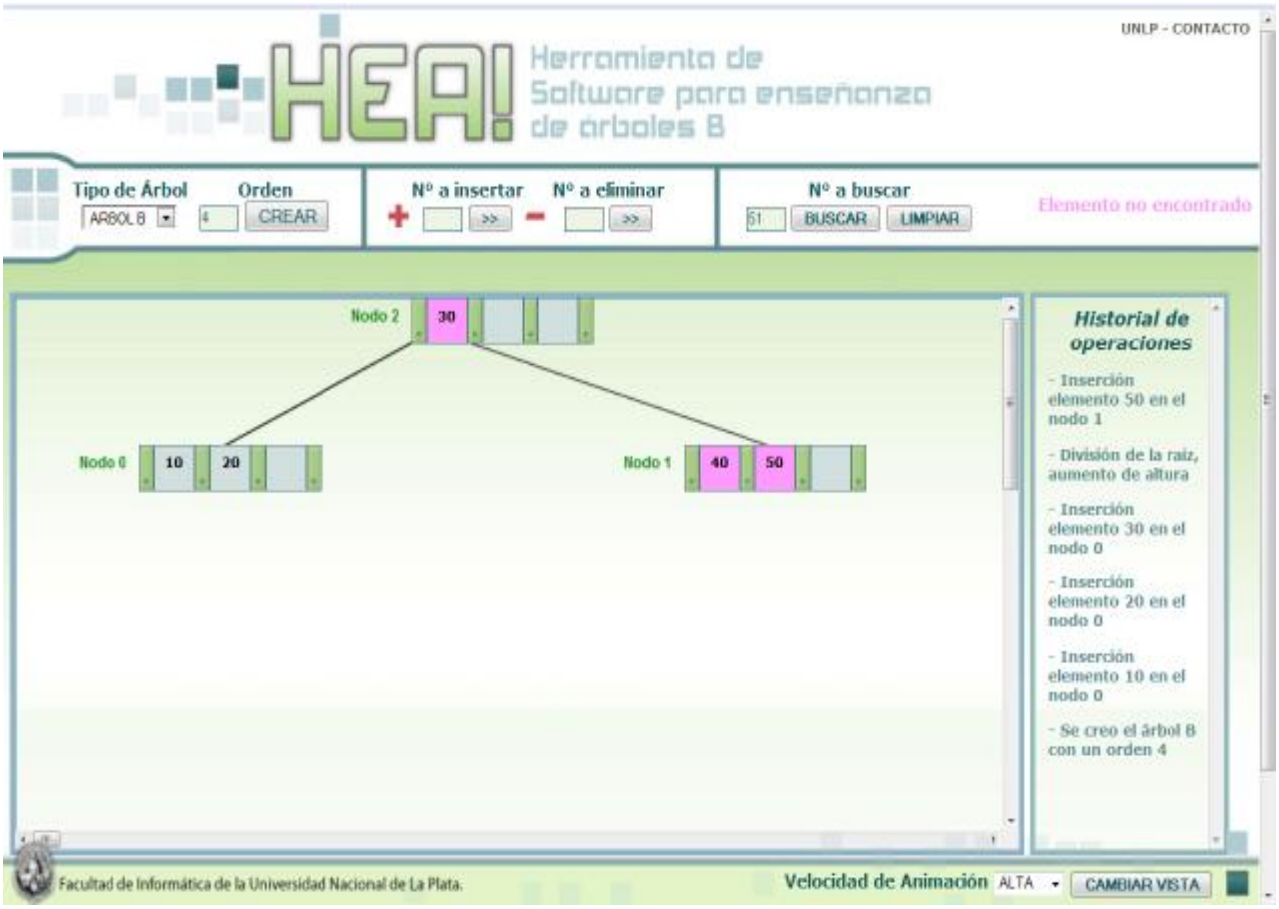

Figura 5.1.6

La figura 5.1.6 describe el caso de búsqueda del elemento 51 el cual no se encuentra en el árbol. En este caso, HEA detalla el camino realizado, coloreando en rosa los puntos recorridos hasta determinar que el elemento buscado no existe en el árbol. También se puede observar la leyenda *Elemento no encontrado* en el costado izquierdo de la zona de búsqueda de la paleta de operaciones.

Más allá de la descripción gráfica a través de animaciones, HEA posee un historial de operaciones (referencia 8 figura 5.1.1). La intención de esta funcionalidad es que queden almacenadas todas las decisiones que el usuario toma sobre el árbol, tanto de configuración como operaciones de inserción, eliminación y búsqueda. De esta manera, desde la creación de la estructura y a medida que se realizan operaciones sobre la misma, se describe en forma de texto toda decisión que el usuario tome como así también la evolución del árbol

en función de su tipo y orden. Esto permite recorrer el historial de las operaciones que se realicen a lo largo de una sesión de aprendizaje. La figura 5.1.7 muestra un ejemplo de un árbol B de orden 4 de dos niveles en donde se puede observar las operaciones realizadas en el historial de operaciones.

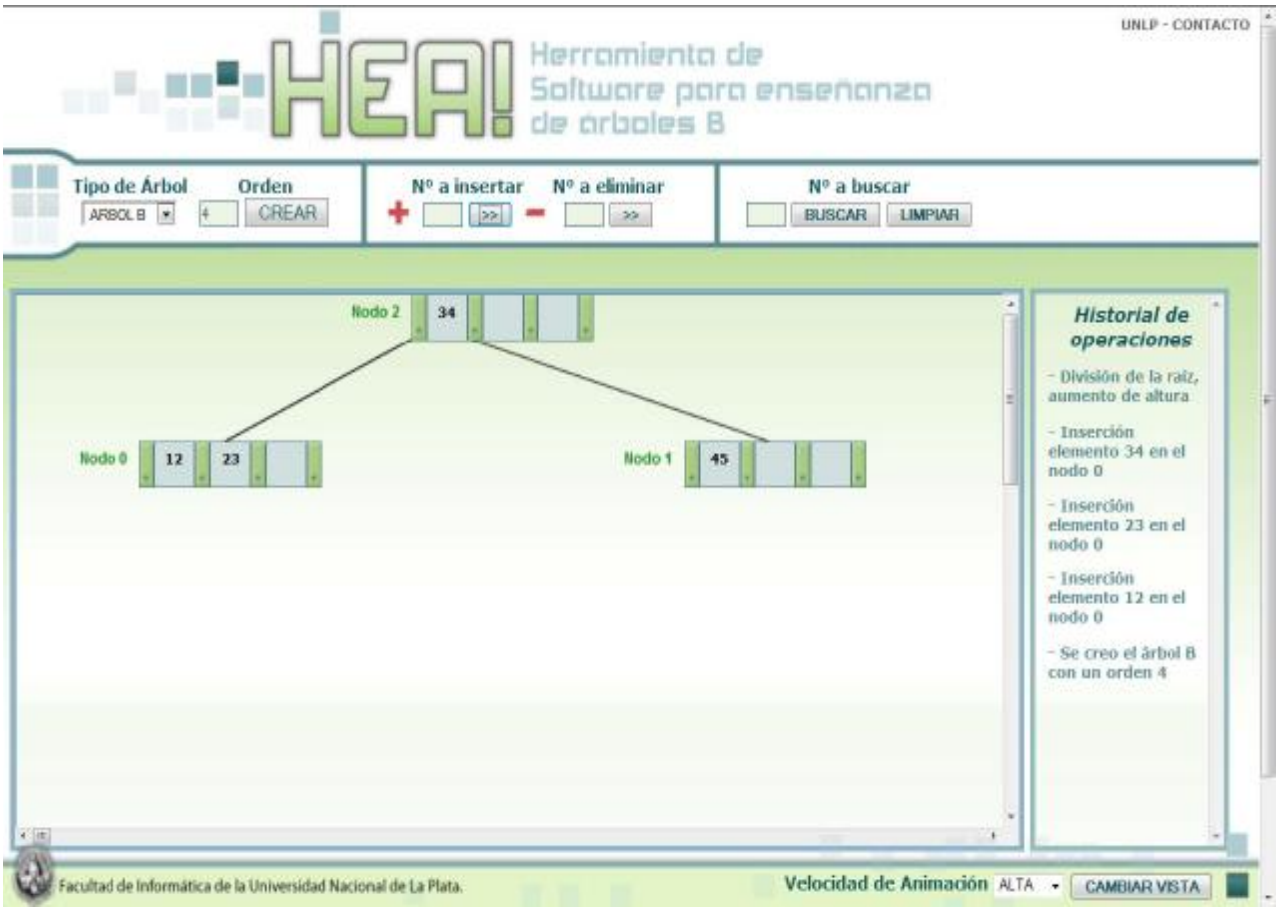

Figura 5.1.7

La finalidad de HEA es brindar la mayor cantidad de posibilidades para que su uso sea fácil y que el alumno enfoque su atención en el aprendizaje de árboles y no se produzcan desconcentraciones. La última operación fue concebida en función a este precepto y consiste en un cambio de visualización (referencia 9 figura 5.1.1).

Como muestra la figura 5.1.8 la herramienta cuenta con dos modos visuales. Por defecto, las animaciones se muestran del lado izquierdo de la pantalla, registrándose en la columna derecha el historial de operaciones. A medida que la estructura crece, el alumno debe utilizar la barra de desplazamiento para

moverse a lo ancho y largo de la misma. Para minimizar la utilización de dicha barra, se podrá optar por una segunda vista que le da más prioridad al área de animación y coloca el historial en la parte inferior de la pantalla.

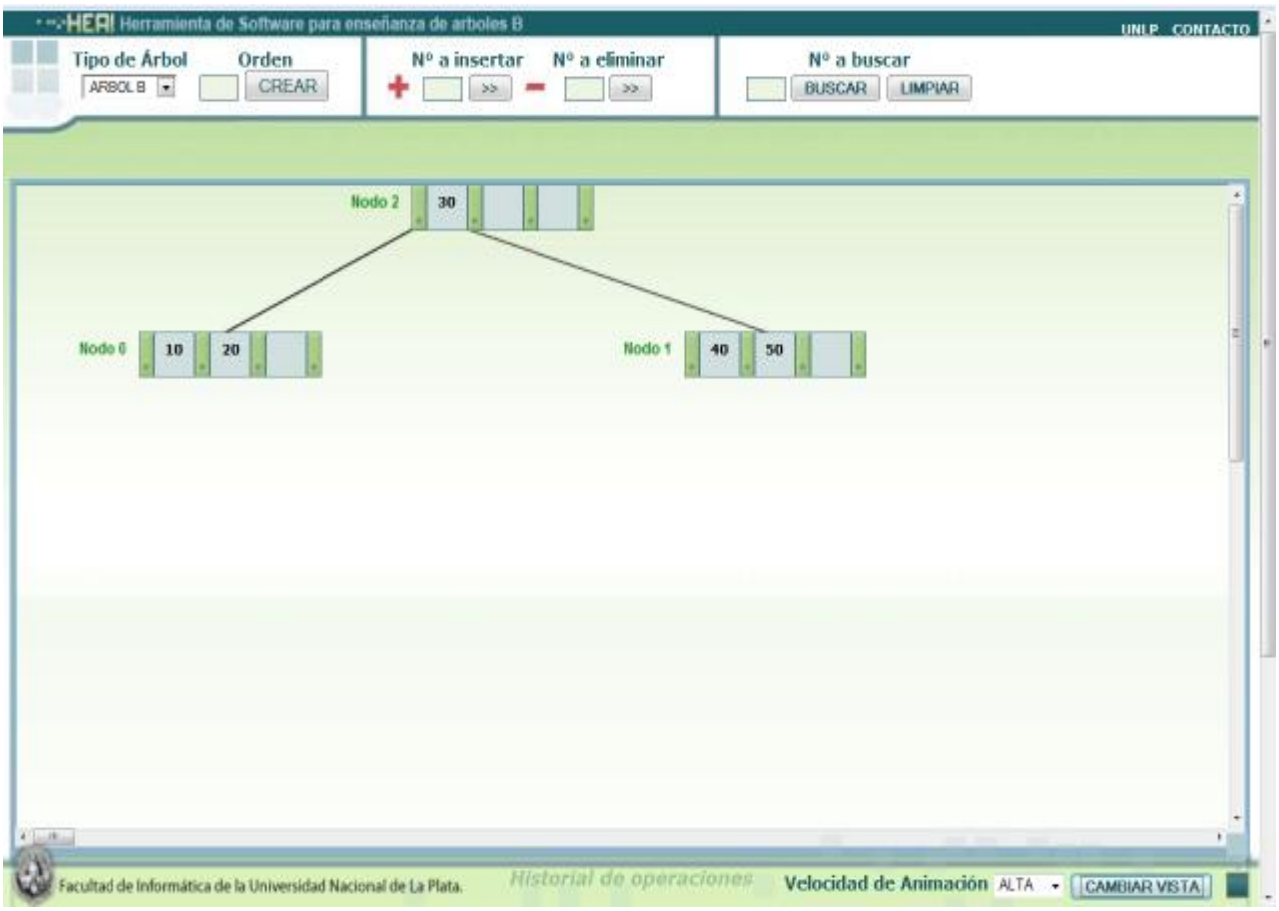

Figura 5.1.8

La figura 5.1.9 muestra cómo reacciona la herramienta cuando es accedida con el navegador Explorer. En los capítulos siguientes se explicará con detalle la tecnología utilizada en la construcción de HEA. Una de ellas es HTML5, en particular la etiqueta canvas, la cual permite dibujar sobre un explorador, utilizando las propiedades del lenguaje de programación javascript. Debido a lo reciente de esta tecnología, el navegador Explorer, aún no está preparado para soportarla correctamente. Si bien se ha anunciado que el navegador Explorer 9 soporta la tecnología antes mencionada, existen incompatibilidades entre los diferentes navegadores en lo que respecta, no solo a HTML5, sino también con propiedades del DOM. Sin perjuicio de esto, la herramienta fue diseñada de

manera tal que si se detecta que el usuario quiere acceder a la misma a través del navegador Explorer se indica un mensaje como el que se muestra a continuación en la figura 5.1.9.

Figura 5.1.9

Tal cual se observa en la figura 5.1.9, HEA detecta que ha sido accedida a través del navegador Explorer por lo que muestra al usuario una serie de links invitándolo a descargarse cualquiera de los navegadores compatibles con la herramienta (Mozilla, Chrome, Opera o Safari).

Por último se presenta la figura 5.1.10 en la cual se muestra un ejemplo de una sesión de aprendizaje avanzada, en la cual se ha creado un árbol B de orden 4 y se han insertado doce claves, haciendo que la estructura tenga tres niveles.

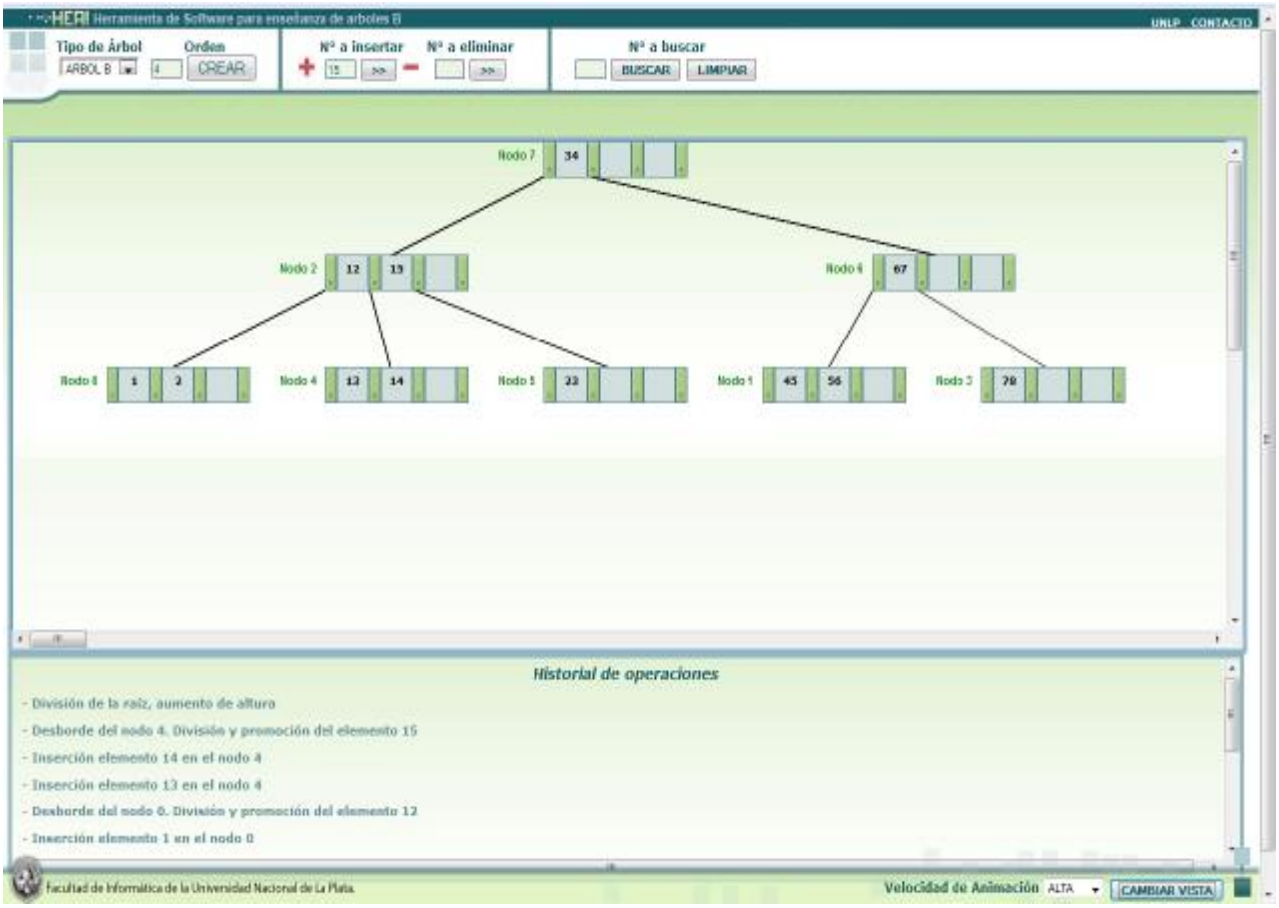

Figura 5.1.10

5.2 Descripción de apariencia y diseño.

La interfaz de HEA ha sido desarrollada para ofrecer una experiencia que se focalice en un uso pedagógico sin perder un entorno amigable al usuario. Para ello se centra en tres estrategias de diseño:

- ✓ Interfaz intuitiva.
- ✓ Presentación amigable.
- ✓ Maleabilidad y dinamismo.

5.2.1 - Interfaz intuitiva

El diseño propone una interfaz donde se despliegan secuencialmente los distintos elementos que la constituyen, ordenados según la lectura occidental de izquierda a derecha, facilitando así realizar los distintos pasos que la creación de estructuras del tipo árbol requiere.

Se busca minimizar los índices de lectura que puedan distraer la atención. Para ello se ha agrupado los pasos, y se ha reducido a cinco zonas donde se muestra el contenido. El encabezado señala una presentación de la herramienta; la zona de operaciones, que reúne la totalidad de la interacción con la creación de árboles, aquella donde se gráfica un modelado del árbol; el historial de operaciones que permite al usuario contar con un seguimiento textual de sus acciones; y una sección que permite agregar operaciones "extras" como el cambio de vista y la configuración de velocidad de las animaciones.

5.2.2 - Presentación amigable

El diseño presenta una visión que permite sentirse en un entorno despojado que invite a concentrarse en la faz educativa. Para ello se recurre al uso de una clave de valor alta, a una paleta cromática donde domina los colores adyacentes (verdes y cian), y reserva el uso de los opuestos (el rojo) para remarcar las operaciones recurrentes. La paleta es fría, y hace uso del degrade hacia la desaturación al tinte, esto facilita la identificación, por parte del usuario, de un espacio neutro que evita la sobrecarga de signos y apuntala la sensación de facilidad de uso. Refuerza la zona activa mediante el cambio dinámico de la paleta.

Así una interfaz amigable propone una relación directa del usuario con el contenido a través de una única página que evita la redundancia y facilita los índices de navegación. Prevalecen las rutinas de interacción que el usuario ya ha incorporado en su uso de la red sin agregar nuevas resoluciones que interfieran en el uso de la herramienta.

5.2.3 - Maleabilidad y dinamismo

La propuesta recurre a una estrategia asincrónica de manera que el usuario no tenga, ante las distintas operaciones, que recargar la página. Fuera de las sabidas ventajas: simplicidad, velocidad de interacción, permanencia del entorno, este uso tiene como principal inconveniente la sensación estática que suele generar. A fin de evitar este problema, y facilitar un aprendizaje dinámico acorde a la cultura multimedia contemporánea, se ha enfatizado gráficamente el carácter procesual de la creación de árboles, mostrando animaciones que describen la evolución de la estructura.

Capítulo 6 - Tecnología utilizada

El objetivo del presente capítulo es describir la tecnología utilizada en la construcción de HEA, como así también cuales fueron los motivos de su elección.

Seleccionar adecuadamente la tecnología que va a formar parte de un proyecto, es lo que comúnmente se denomina arquitectura del software, y cumple un papel fundamental a la hora de determinar el éxito o fracaso de un sistema.

6.1 Elección de la Tecnología

Al momento de analizar que tecnología utilizar para la construcción de HEA, hubo dos puntos que se tuvieron en cuenta: la facilidad de acceso y las animaciones, a través de las cuales, se muestra la evolución de las estructuras del tipo árbol.

Es innegable que en los últimos años la tecnología web ha avanzado notoriamente, y que su utilización, hace que la información y las herramientas, estén cada vez más al alcance de la mano de los usuarios comunes. Nada parece desplazar a la web como mayor fuente de información y consulta. Frente a este escenario, uno de los objetivos planteados fue que HEA pueda ser accedido vía web. En función de esto las preguntas que surgen son: ¿todos los posibles usuarios de HEA tienen acceso a internet?, ¿es justo que aquellos usuarios que no poseen conexión no tengan acceso a HEA? Estos interrogantes motivaron a pensar en HEA como un híbrido, es decir, que aquellos usuarios que tuviesen internet accedieran utilizando la red mientras que los que no, pudiesen instalar la herramienta en su computadora y utilizarla sin mayores inconvenientes.

En pos de cumplir con lo anteriormente mencionado, se planteó como alternativa, que HEA sea construido íntegramente con tecnología web, pero que las mismas se ejecuten del lado del cliente. Resulta necesario aclarar al lector cual es la diferencia entre tecnología que se ejecuta del lado del cliente y la que lo hace del lado del servidor. En la tecnología web que se ejecuta del lado del cliente, se dice que todo el procesamiento lo realiza la máquina donde se está ejecutando la aplicación. Una definición más formal sería que la ejecución de los programas o scripts se realiza en el navegador del usuario, es decir en la PC del usuario. Por el contrario, que una tecnología se ejecute del lado del servidor, significa que el navegador del cliente hace solicitudes a un servidor en el cual se realiza el procesamiento, y se devuelve una respuesta al navegador, para que este se encargue de mostrarla.

Las figuras 6.1.1 y 6.1.2 muestran el camino realizado al momento de crear páginas con lenguajes que se ejecutan del lado del cliente y del lado del servidor respectivamente. Existe una vasta bibliografía que explica el paradigma cliente-servidor, pero no es objeto de este apartado ahondar en el tema.

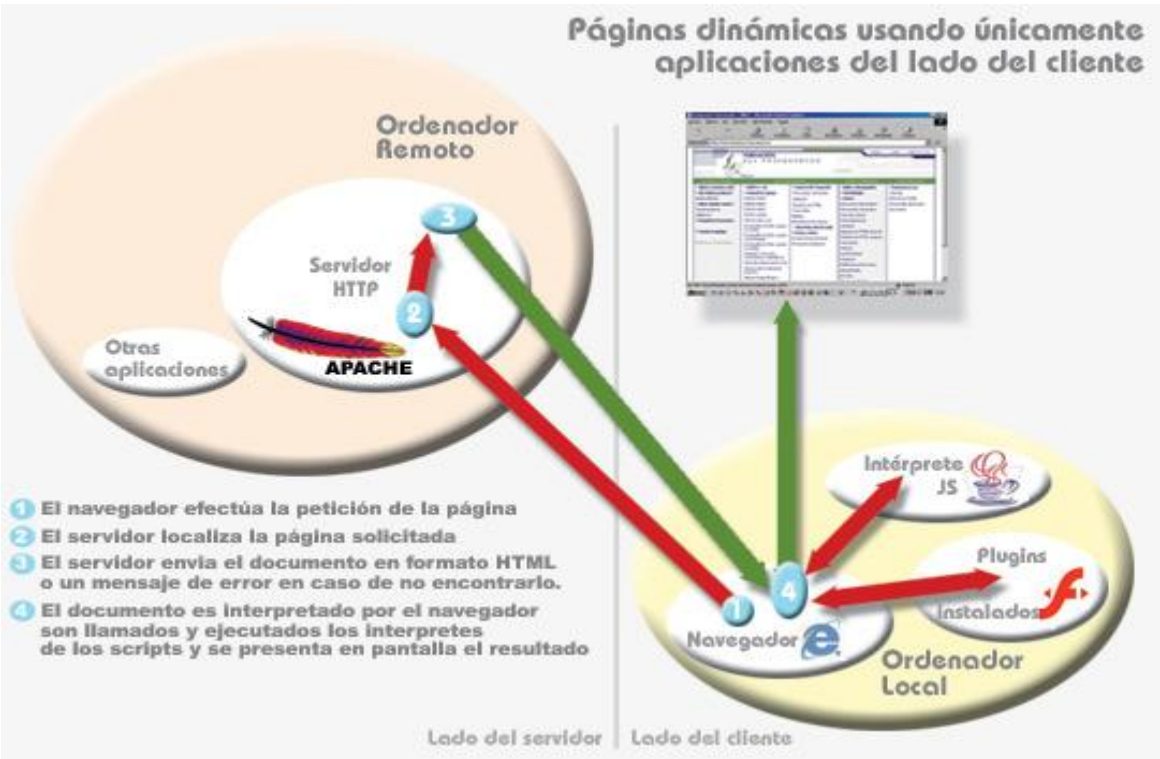

Figura 6.1.1 [11]

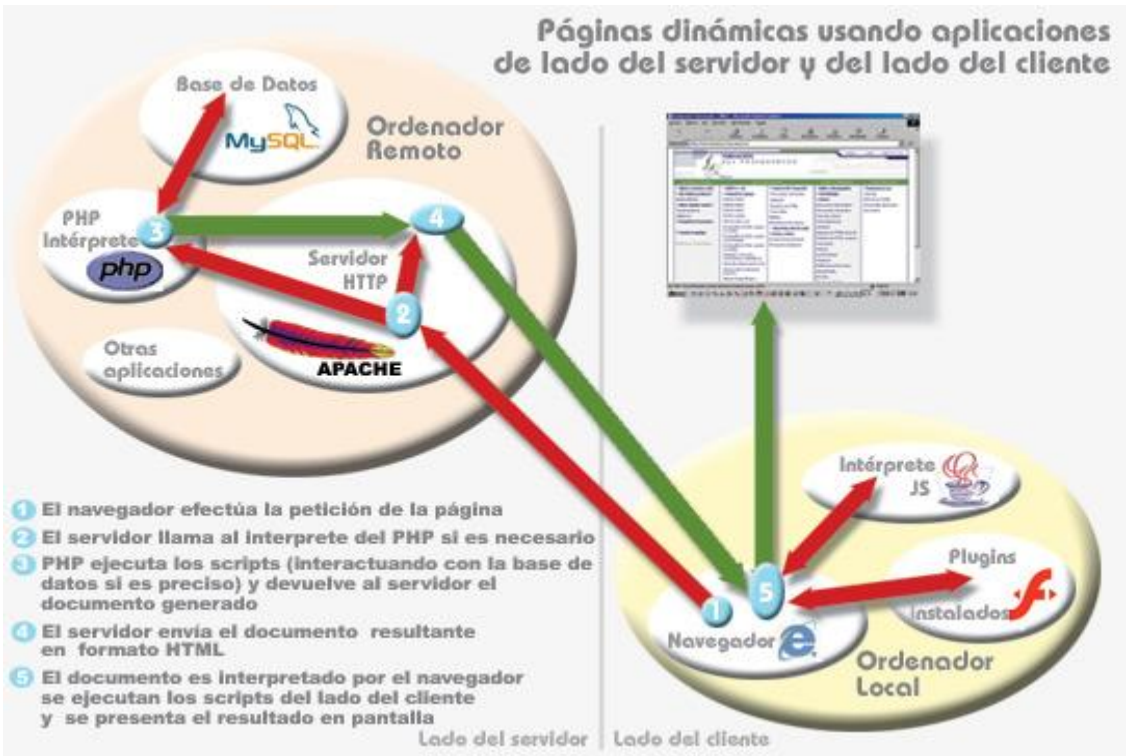

Figura 6.1.2 [11]

En definitiva, desarrollar la herramienta con tecnología web que se ejecute del lado del cliente, permite que HEA sea alocado en un servidor y que pueda ser accedido a través de internet, aunque el procesamiento se realice en el cliente; o que el usuario se lo descargue en su computadora y pueda utilizarlo sin más requerimientos que tener un navegador web instalado en su computadora.

Otro de los puntos a considerar para la elección de la tecnología utilizada fue contar con la posibilidad de realizar animaciones que facilitaran el aprendizaje de árboles, y que la utilización de HEA no provocara una carga extra para el alumno, sino por el contrario, que éste se centrara en el aprendizaje de árboles sin tener que poseer conocimientos previos acerca de la utilización de la herramienta.

La librería Jquery, la cual será explicada en detalle más adelante, permite la realización de animaciones y cumple con todos los objetivos propuestos ya que por estar implementada íntegramente en el lenguaje de programación JavaScript, se ejecuta del lado del cliente, cumpliendo también así, el primer punto planteado.

En conclusión, HEA es una página web, escrita en HTML, con JavaScript embebido, y utilizando como complemento la librería Jquery y una etiqueta especial de HTML5 denominada canvas, que permite realizar gráficos sobre un navegador WEB. [12]

6.2 Descripción de la Tecnología

La finalidad de este apartado es describir muy brevemente la tecnología utilizada en la construcción de HEA. Es importante aclarar que la intención es que el lector posea una idea de las herramientas utilizadas, sin profundizar cada una de ellas, lo que excede el objeto de este trabajo.

6.2.2 - HTML

HTML, Hyper Text Markup Language (Lenguaje de marcación de Hipertexto) es el lenguaje utilizado comúnmente en la www (World Wide Web). Fue concebido

por el físico nuclear Tim Berners-Lee, quien tomo como base dos herramientas: el concepto de hipertexto, el cual permite conectar dos elementos entre sí, y el SGML(Lenguaje Estándar de Marcación General) que sirve para colocar marcas o etiquetas en un texto.

HTML por sí mismo no es un lenguaje de programación sino que es un sistema de etiquetas y no posee un compilador, razón por la cual, los errores de sintaxis no se detectan y se visualizarán como éste lo entienda.

Para crear un documento HTML, solo es necesario un procesador de texto, sobre el cual se escribirán una serie de etiquetas generándose un archivo con extensión HTML que podrá ser visualizado en cualquier navegador web, como puede ser Mozzilla, Explorer, Chrome, Opera, etc. [13]

6.2.3 - HTML 5

En la construcción de HEA, como ya se mencionará oportunamente, se utiliza una etiqueta denominada canvas. Esta etiqueta forma parte de una nueva especificación de HTML, denominada HTML5. A continuación de describen algunos detalles de esta etiqueta y de la nueva especificación.

HTML 5 proporciona una plataforma para desarrollar aplicaciones web más parecidas a las aplicaciones de escritorio, donde su ejecución dentro de un navegador no implique falta de recursos o facilidades para resolver las necesidades reales de los desarrolladores. Para ello se han creado unas APIs (interface de programación de aplicaciones) que permitan trabajar con cualquiera de los elementos de la página y realizar acciones que hasta hoy era necesario realizar por medio de tecnologías accesorias.

Entre las ventajas que presenta HTML 5 encontramos:

- ✓ *Estructura del cuerpo*: La mayoría de las webs tienen un formato común, formado por elementos como cabecera, pie, navegadores, etc. HTML 5 permite agrupar todas estas partes de una web en nuevas etiquetas que representarán cada uno de las partes típicas de una página.
- ✓ *Etiquetas para contenido específico*: Hasta ahora se utilizaba una única etiqueta para incorporar diversos tipos de contenido enriquecido, como animaciones Flash o vídeo. Ahora se utilizarán etiquetas específicas para cada tipo de contenido en particular, como audio, vídeo, etc.
- ✓ *Canvas*: es un nuevo componente que permite dibujar, por medio de las funciones de un API, en la página todo tipo de formas, que pueden estar animadas y responder a interacción del usuario. Es algo así como las posibilidades que nos ofrece Flash, pero dentro de la especificación del HTML y sin la necesidad de tener instalado algún plugin.
- ✓ *Bases de datos locales*: el navegador permite el uso de una base de datos local, con la que se puede trabajar en una página web por medio del cliente y a través de un API. Es similar a las Cookies, las cuales son un fragmento de información que se almacena en el disco duro del

visitante de una página WEB la cual puede ser recuperada en futuras sesiones permitiendo tener un registro de las visitas hechas por un usuario, pero pensadas para almacenar grandes cantidades de información, lo que permite la creación de aplicaciones web que funcionen sin necesidad de estar conectados a Internet.

- ✓ *Web Workers*: son procesos que requieren bastante tiempo de procesamiento por parte del navegador, pero que se pueden realizar en un segundo plano, para que el usuario no tenga que esperar que se terminen para empezar a usar la página. Para ello se dispone también de un API para el trabajo con los Web Workers.
- ✓ *Geolocalización*: Las páginas web se pueden localizar geográficamente por medio de un API que permite la Geolocalización.
- ✓ *Nuevas APIs para interfaz de usuario*: temas tan utilizados como el "drag & drop" (arrastrar y soltar) en las interfaces de usuario de los programas convencionales, son incorporadas al HTML 5 por medio de un API.
- ✓ *Fin de las etiquetas de presentación*: todas las etiquetas que tienen que ver con la presentación del documento, es decir, que modifican estilos de la página, son eliminadas. La responsabilidad de definir el aspecto de una web correrá a cargo únicamente de CSS. [14]

6.2.4 - JavaScript

JavaScript es un lenguaje de programación que se ejecuta del lado del cliente, porque es el navegador el que soporta *la carga de procesamiento.* Debido *a* su compatibilidad con la mayoría de los navegadores modernos, es el lenguaje de programación más utilizado que se ejecuta del lado del cliente.

Gracias a JavaScript es posible crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones Javascript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, y tal vez el único, con que cuenta este lenguaje es el propio navegador.

Se dice que este lenguaje de programación es el siguiente paso, después del HTML, que puede dar un desarrollador que decida mejorar sus páginas y la potencia de sus proyectos.

Las posibilidades que brinda javaScript al desarrollador, pueden separarse en dos ramas. Por un lado permite la realización de efectos especiales sobre páginas web, para crear contenidos dinámicos y elementos de la página que tengan movimiento, cambien de color o cualquier otro dinamismo. La segunda rama permite ejecutar instrucciones como respuesta a las acciones del usuario, con lo que podemos crear páginas interactivas. Estas dos vertientes son muy utilizadas en el desarrollo de HEA.

JavaScript es un lenguaje con muchas posibilidades, permite la programación de pequeños scripts, pero también de programas más grandes, orientados a objetos, con funciones, estructuras de datos complejas, etc. Además, Javascript pone a disposición del programador todos los elementos que forman la página web, para que éste pueda acceder a ellos y modificarlos dinámicamente.

6.2.5 - Jquery

A la hora de definir Jquery, se puede decir que es un framework Javascript, pero quizás muchos se preguntarán qué es un framework. Un framework es un producto que sirve como base para la programación avanzada de aplicaciones, que aporta una serie de funciones o códigos para realizar tareas habituales. Para decirlo de otra manera, framework es un conjunto de librerías de código que contienen procesos o rutinas ya listos para usar. La utilización de frameworks permite no tener que desarrollar las tareas más básicas, puesto que en el propio framework ya hay implementaciones que están probadas, funcionan y no se necesitan volver a programar haciendo eco de un precepto fundamental en la programación como es la reusabilidad.

En el caso de jQuery, es un framework para el lenguaje Javascript, luego será un producto que simplificará notoriamente al programar en este lenguaje.

Así pues, este framework Javascript, ofrece una infraestructura con la que se dispone mucha mayor facilidad para la creación de aplicaciones complejas del lado del cliente. Por ejemplo, con jQuery se obtiene ayuda en la creación de interfaces de usuario, efectos dinámicos, aplicaciones que hacen uso de Ajax, etc. Este framework pone a disposición una interfaz para programación que permite hacer tareas con el navegador que funcionarán para todos los visitantes. Simplemente se debe conocer las librerías del framework y programar utilizando las clases, sus propiedades y métodos para la consecución de nuestros objetivos.

El archivo del framework ocupa unos 56 KB, lo que es razonable y no retrasará mucho la carga de la página (si el servidor envía los datos comprimidos, lo que es bastante normal, el tamaño de jQuery será de unos 19 KB). Además, el servidor lo enviará al cliente la primera vez que visite una página del sitio. En siguientes páginas, el cliente ya tendrá el archivo del framework, por lo que no necesitará transferirlo y lo tomará de la caché. Con lo que la carga de la página sólo se verá afectada por el tamaño de este framework una vez por usuario. [15]

Es importante aclarar que la tecnología utilizada en la construcción de HEA es de uso libre, esto significa que pueden ser utilizadas sin la necesidad de contar con algún tipo de licencia.

Capítulo 7 - Conclusiones y Trabajos Futuros

7.1 - Conclusiones

Uno de los aspectos más interesantes que tiene la informática y en particular el desarrollo de software, es la posibilidad de enfrentarse constantemente con nuevos problemas. Uno de los desafíos con los que se debe enfrentar un buen diseñador de software es el de resolver problemas de manera eficiente. Sin embargo, HEA cuenta con una particularidad, debe ser útil como complemento educativo. Desarrollar una herramienta con este propósito, reforzó la importancia de realizar uno de los primeros pasos del ciclo de vida de desarrollo de software, el análisis de requerimientos, considerando para esto, no solo aspectos funcionales sino también académicos.

La importancia de aprovechar los avances tecnológicos aplicados al proceso enseñanza-aprendizaje es otra característica a destacar, desarrollando software que complemente este proceso, sin dejar de lado un precepto enunciado en capítulos anteriores: el mejor profesor no es aquel que da las mejores respuestas a sus alumnos sino aquel que ayuda a éstos a que las encuentren.

Otra de las reflexiones a partir de este proyecto tiene que ver con el cuidado del proceso de enseñanza-aprendizaje. Las actividades de enseñanza que realizan los profesores están inevitablemente unidas a los procesos de aprendizaje que, siguiendo sus indicaciones, realizan los estudiantes. El objetivo del docente siempre consiste en el logro de determinados objetivos educativos y la clave del éxito está en que los estudiantes puedan y quieran realizar las operaciones cognitivas convenientes para ello, interactuando adecuadamente con los recursos educativos a su alcance. [16]. Por ende los recursos educativos con los que dispongan los profesores al momento de trasmitir conocimientos juega un papel fundamental, pero sin perjuicio de ello, se debe prestar especial atención en las herramientas utilizadas, con la

finalidad de que el razonamiento, el descubrimiento, la constancia y la motivación nunca dejen de ser los principales preceptos que deben mantenerse en el proceso de aprendizaje.

La complejidad en el desarrollo de una herramienta visual y en particular la realización de animaciones fue un problema complejo que derivó en la investigación de tecnologías adecuadas para la programación, tal como se mostró en el capítulo anterior, y en metodologías de desarrollo que permitieran no solo la resolución del problema sino también la mejor organización y escalabilidad del proyecto. De esta manera el desarrollo en capas fue de gran utilidad.

HEA fue desarrollado siguiendo un patrón para la arquitectura de software denominado MVC. El patrón MVC (modelo, vista, controlador) separa los datos de una aplicación, la interface del usuario y la lógica de control en tres capas diferentes. Esto permite que cada capa se encargue de una tarea específica, logrando independencia entre ellas, y donde la modificación de una de las capas no afecta el funcionamiento de las otras logrando, no solo organizar el código correctamente, sino también hacer que la escalabilidad del mismo resulte mucho más fácil.

El MVC fue de gran utilidad para el desarrollo de HEA. Siguiendo el esquema planteado por este patrón, la arquitectura de software de HEA está planteada en tres capas. La primera capa (modelo) concentra la lógica de resolución de árboles. Esta primer capa se comunica con la segunda (controlador), la cual es encargada de encolar operaciones, y por último la tercer capa (vista) es la que tiene como tarea la presentación de las animaciones comunicándose con el controlador, el cual provee las operaciones a mostrar.

Resulta necesario poner especial atención en la última capa (vista) la cual concentra gran parte de la complejidad del desarrollo. La realización de animaciones gráficas fue un gran desafío en el desarrollo de HEA. La resolución de este problema derivo en dos conclusiones que se detallan a continuación.

En una primera instancia, el desarrollo de animaciones plantea el problema del posicionamiento de los gráficos en la pantalla. Determinar las coordenadas de ubicación en pixeles de los nodos componentes de un árbol, teniendo en cuenta no solo la ubicación propia de un nodo, sino también la posición de los nodos hermanos y del nodo padre fue un gran problema desde el punto de vista matemático. Es en este punto en donde se plantea la importancia de tener una base matemática para la resolución de problemas, lo que ayudó a la identificación de patrones comunes para poder obtener una única función matemática que permitiese el reacomodamiento del árbol, logrando obtener la ubicación exacta de cada nodo en función de la ubicación del nodo padre, los nodos hermanos y los nodos hijos, y que dicha función sirviese tanto para cuando el árbol tiene un solo nivel como para cuando tiene más de uno.

En segundo lugar se destaca la importancia de la programación recursiva. La aplicación de este concepto fue de gran utilidad debido a que muchas operaciones realizadas sobre árboles son de naturaleza recursiva ya que se repiten a lo largo de la estructura hasta llegar a la raíz, momento en el que se alcanza el caso base. La utilización de esta técnica de programación ha sido de gran ayuda dado que permite la reducción de código y algoritmos más legibles.

En definitiva, la complejidad del problema gráfico derivó en la necesidad de aplicar diferentes conceptos. Debido a esto es que la mayor parte del esfuerzo se centró en las animaciones, razón por la cual, se plantean como trabajos futuros una serie de funcionalidades las cuales son descriptas en el próximo apartado.

Hasta el momento, HEA fue testeado por un grupo de docentes de la cátedra Introducción a las Bases de Datos de la Facultad de Informática. El testeo realizado se puede dividir en dos etapas: la primera etapa tiene que ver con aspectos funcionales, en la que los resultados obtenidos han sido altamente satisfactorios.. En esta etapa se ha probado HEA en los navegadores: Explorer, Mozilla, Opera, Chrome y Safari, y en todos, excepto Explorer, funciona sin ningún tipo de dificultad. El problema en el navegador Explorer, como se ha explicado en el capítulo anterior, radica en que en las versiones actuales, aún

no ha sido incorporada la posibilidad de soportar HTML5. Frente a esta dificultad se ha realizado un sondeo entre los posibles usuarios de la herramienta y se ha detectado que la mayoría utiliza el navegador Mozilla y Chrome.

La segunda parte del testeo tiene que ver con aspectos visuales de la herramienta; en esta etapa se ha mejorado la herramienta en lo que respecta a disposición de la paleta de operaciones, animaciones y el historial de acciones, modificando también colores y explicaciones, teniendo en cuenta la experiencia de los docentes frente a la necesidad de los alumnos, futuros usuarios de HEA.

Por último se estima que HEA será utilizado por más de 500 alumnos por año pertenecientes a la carreras de Licenciatura en Sistemas, Licenciatura en Informática y Analista Programador Universitario de la Facultad de Informática de la Universidad Nacional de La Plata, permitiendo de esta manera seguir avanzando en el proyecto a partir de la experiencia que se obtenga y del estudio de usabilidad que se prevé realizar.

7.2 - Trabajos futuros

Uno de los trabajos prioritarios a realizar en HEA tiene que ver con la confección de un archivo de carga. Este trabajo consiste en que el usuario de la herramienta pueda confeccionar un archivo de texto con las claves que quiera insertar, eliminar o buscar en el árbol. De esta manera, y utilizando un estándar de notación, el usuario puede crear dicho archivo y la herramienta permite la carga del mismo. Una vez cargado el archivo, HEA comienza a leerlo mostrando la evolución de la estructura a través de animaciones. Un posible estándar de notación es +clave (inserción de una clave), -clave (eliminación de una clave), *clave (búsqueda de una clave). De esta manera una posible secuencia de un archivo de carga podría ser: +100,+10,+3,-10,*100 frente a la cual HEA mostraría con animaciones la evolución del árbol con la siguiente

secuencia de operaciones: inserción del 100, inserción del 10, inserción del 3, eliminación del 10 y búsqueda del 100.

Otro trabajo previsto consiste en la incorporación de descripciones a través de archivos de audio. De esta manera el alumno contará no solo con las explicaciones textuales que se muestran en el historial de operaciones, sino que también a medida que se realicen operaciones sobre un árbol se presentarán explicaciones sonoras. La idea consiste en que el usuario cuente con la posibilidad de activar o desactivar cuando lo desee, el modo de explicación auditiva.

Contar con la posibilidad de poder guardar un trabajo, también se ha planteado como tarea futura. Esto consiste en que el usuario de HEA pueda almacenar en un archivo el trabajo realizado en un momento dado. Una de las posibilidades es que el archivo se guarde en formato XML.

XML es un metalenguaje extensible de etiquetas que busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de trozos de información. De esta forma se podría guardar el trabajo realizado con HEA, de manera tal de que el alumno en una futura sesión de aprendizaje pueda continuar a partir del punto almacenado en una sesión anterior. Además esta funcionalidad brindará la posibilidad de que los alumnos intercambien trabajos en caso de ser necesario.

Por último, la posibilidad de imprimir el árbol que el alumno va visualizando a través de las animaciones es una funcionalidad deseada a futuro. De esta manera se permitirá tener en soporte papel, el árbol sobre el que ha trabajado para futuros repasos o consultas prácticas.

Referencias

[1] Definición de motivación [internet]. [Citado 2010 Nov. 16]; Disponible desde: http://definicion.de/motivacion/

[2] Slady: Java B-Tree applet animation [Internet]. [Citado 2011 Abr. 2]; Disponible desde: http://slady.net/java/bt/view.php

[3] Animación del Método de Borrado [Internet]. [Citado 2011 Abr. 2]; Disponible desde: http://usuarios.multimania.es/arbolesbpro/animaborrado.htm

[4] Animación del Método de Inserción [Internet]. [Citado 2011 Abr. 2]; Disponible desde: http://usuarios.multimania.es/arbolesbpro/animainsercion02.htm

[5] Animación del Método de Búsqueda [Internet]. [Citado 2011 Abr. 2]; Disponible desde: http://usuarios.multimania.es/arbolesbpro/animabusca.htm

[6] Universidad de La Serena. Simulación en Java [Internet]. [Citado 2011 Abr. 2]; Disponible desde: http://usuarios.multimania.es/arbolesbpro/aplicacion/ArbolB.html

[7] Ley Federal de Educación. [Internet]. [Citado 2010 Oct. 3]; Disponible desde: http://www.jusneuquen.gov.ar/share/legislacion/leyes/leyes_nacionales/ley_24195.htm

[8] Rodolfo Bertone, Pablo Thomas. Introducción a las Bases de Datos Fundamentos y Diseño. Editorial Pearson. ISBN 978-987615136-8. Primera Edición. Mayo 2011

[9] Base de Datos [Internet]. [Citado 2010 Nov. 20]; Disponible desde: http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c3/c3.htm

[10] Michael J. Folk , Bill Zoellick. Estructuras de archivos - Un conjunto de herramientas conceptuales.

[11] Lenguajes del lado servidor o cliente [Internet]. [Citado 2011 Ene. 5]; Disponible desde: http://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servi

dor_o_cliente.html

[12] Revista La Tarea - La Tecnología aplicada a la educación/Priscila Ramírez Sandoval y Francisco Javier Vargas Rangel [Internet]. [Citado 2010 Nov. 19]; Disponible desde: http://www.latarea.com.mx/articu/articu12/remisa12.htm

[13] Especificación HTML 4.01 [Internet]. [Citado 2011 Ene. 5]; Disponible desde: http://html.conclase.net/w3c/html401-es/cover.html

[14] HTML5 [Internet]. [Citado 2011 Ene. 5]; Disponible desde: http://dev.w3.org/html5/spec/Overview.html

[15] jQuery: The Write Less, Do More, JavaScript Library [Internet]. [Citado 2011 Ene. 5]; Disponible desde: http://jquery.com/

[16] Proceso de Enseñanza y Aprendizaje [Internet]. [Citado 2011 Ene. 12]; Disponible desde: http://peremarques.pangea.org/actodid.htm