UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN – TACNA

Escuela de Posgrado

MAESTRÍA EN COMPUTACIÓN E INFORMÁTICA

VELOCIDAD DE RESPUESTA DE LOS ALGORITMOS DE BÚSQUEDA DE DATOS CONTENIDOS EN ESTRUCTURAS ESTÁTICAS Y DINÁMICAS

TESIS

PRESENTADA POR:

ING. EDWIN ANTONIO HINOJOSA RAMOS

Para optar el Grado Académico de:

MAESTRO EN CIENCIAS (*MAGISTER SCIENTIAE*)
CON MENCIÓN EN COMPUTACIÓN E INFORMÁTICA

TACNA – PERÚ

2014

UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN – TACNA

Escuela de Posgrado

MAESTRÍA EN COMPUTACIÓN E INFORMÁTICA

VELOCIDAD DE RESPUESTA DE LOS ALGORITMOS DE BÚSQUEDA DE DATOS CONTENIDOS EN ESTRUCTURAS ESTÁTICAS Y DINÁMICAS

Tesis sustentada y aprobada el 18 de diciembre del 2014; estando el jurado calificador integrado por:

PRESIDENTE : .....
Dr. Heber Melbin Cabrera Cruz

SECRETARIO : .....
M.Sc. Wilder Roger Miñano León

MIEMBRO : .....
M.Sc. Edgar Aurelio Taya Acosta

ASESOR : .....
Dr. Julio Miguel Fernández Prado

AGRADECIMIENTOS

Mi agradecimiento eterno a Dios por guiarme y protegerme en la vida. A mi Familia, por inspirar en mí los más puros sentimientos de amor fraternal. A Giovanita por ser mi gran amor y quien me brinda su apoyo incondicional.

A mis Maestros, por compartir sus conocimientos y su sabiduría. A todos quienes me apoyaron de una u otra forma para la culminación de la presente investigación.

DEDICATORIA

A mi Padre que descansa en el Reino de Dios, a mi Madre por acompañarme siempre. A mis Padres que me dieron la Vida y me formaron para actuar con rectitud y honestidad.

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Los datos almacenados en estructuras de datos dinámicas del tipo Árbol AVL permiten obtener mayor velocidad de respuesta en las operaciones de búsqueda de datos específicos en comparación con las estructuras de datos estáticas del tipo Array unidimensional. Esta velocidad está en función del número de comparaciones efectuadas en el proceso de búsqueda y claramente se verifica que el número de comparaciones efectuada en los arreglos es mucho mayor que las comparaciones efectuadas en el Árbol AVL cuando se realiza el proceso de localización de claves. Pero también se observa que el tiempo que se tarda en insertar las claves en un Array unidimensional es mucho menor que el tiempo requerido para almacenar los datos en un Árbol AVL. Pero, en promedio, la velocidad de respuesta de los algoritmos de búsqueda de datos contenidos en estructuras dinámicas del tipo Árbol AVL es mayor que la velocidad de respuesta cuando el proceso de búsqueda se efectúa en las estructuras estáticas del tipo Array unidimensional.

ABSTRACT

The data stored in dynamic data structures of type AVL tree enable higher response rate lookup specific data than static structures dimensional Array data type. This speed depends on the number of comparisons performed in the search process and clearly verified that the number of comparisons made in the arrangements is greater than the comparisons made in the AVL tree when the key location process is performed. But it is also observed that the time it takes to insert the key in a one-dimensional Array is much less than the time required to store the data in a AVL tree. But, on average, the response speed of the search algorithms dynamic data in AVL tree-like structures is larger than the response speed when the search process is performed in the static-dimensional Array type structures.

INTRODUCCIÓN

En todo proceso computacional es necesaria la operación de datos que pueden estar contenidos en memoria interna o en memoria externa. Existen diferentes organizaciones lógicas de los datos a las que denominamos estructuras de datos. Cada una de ellas posee diferentes características que hacen que sean más o menos ventajosas para diferentes procesos en el ordenador. El objetivo fundamental de toda estructura de datos es la organización de los datos en la memoria del ordenador para poder operar luego esos datos en tiempo de ejecución de los programas.

Existen dos grupos bien diferenciados de estructuras de datos. Las estructuras estáticas y las estructuras dinámicas. Las primeras no pueden variar su tamaño y capacidad de almacenamiento en tiempo de ejecución lo que representa un riesgo de desbordamiento cuando la cantidad de datos supera a su capacidad de almacenamiento. En contraste, las estructuras dinámicas pueden modificar su capacidad de almacenamiento de datos en tiempo de ejecución, con lo cual se consume sólo el espacio de memoria necesario que requiere la aplicación. Con esto se elimina el riesgo de desbordamiento de la estructura. Pero es importante señalar

que la complejidad en el diseño de los algoritmos que operan datos contenidos en estructuras dinámicas es alta y el tiempo de desarrollo e implementación de los algoritmos aumenta considerablemente.

En el presente trabajo de investigación se estudia la diferencia de dos estructuras de datos, una estática que es el Array unidimensional y otra estructura dinámica que es el Árbol AVL. Se estudia el tiempo de respuesta para los procesos de búsqueda de datos contenidos en estas estructuras y con ello determinar la velocidad de respuesta de los algoritmos de búsqueda que actúan sobre estas estructuras. Para ello se implementó en el lenguaje C++ un programa que genere en forma automática los datos y los almacene en las dos estructuras y a partir de ello efectuar operaciones de búsqueda cronometrando el tiempo de respuesta de los procesos para evaluar su desempeño.

CAPÍTULO I

PLANTEAMIENTO DE LA INVESTIGACIÓN

1.1. Planteamiento del problema

Los grandes volúmenes de datos que se generan día a día requieren dispositivos de almacenamiento masivo de datos de gran capacidad y equipos de cómputo de alta potencia de cálculo que permita reducir los tiempos de acceso y recuperación de datos para su explotación en aplicaciones computacionales.

Pero, además de lo señalado en el párrafo anterior, es muy importante la forma cómo se estructuran los datos en memoria. Podemos tener equipos computacionales de alta potencia de cálculo y dispositivos de almacenamiento masivo de datos, pero si los datos no son estructurados apropiadamente en memoria, no se alcanzarán altas velocidades de respuesta en los procesos de búsqueda de datos almacenados en el ordenador.

Todo ello me lleva a plantear la siguiente interrogante:

¿Cómo influyen las estructuras de datos en la velocidad de respuesta de los algoritmos de búsqueda en memoria del ordenador?

1.2. Justificación e importancia de la investigación

Los sistemas computacionales necesitan de datos que luego se transformarán en información mediante algoritmos computacionales según un propósito específico. Los datos deben estructurarse en dispositivos de almacenamiento externo y para su procesamiento deben extraerse los datos de la memoria externa y estructurarse en memoria interna del ordenador y es allí donde se efectúan los procesos computacionales sobre los datos.

Una de las operaciones que se debe efectuar en todo momento es la búsqueda de los datos en memoria interna o externa. La velocidad de respuesta de los algoritmos de búsqueda es fundamental en la búsqueda de la eficiencia de los procesos computacionales.

Las estructuras que se utilicen para contener los datos en memoria son importantes para poder alcanzar diferentes velocidades de localización de los datos buscados. Por ello es importante efectuar el estudio de los algoritmos de búsqueda aplicados a datos contenidos en estructuras estáticas y dinámicas y comparar cuál de estas estructuras generan mayores velocidades de respuesta. Esto servirá de base para el diseño de gestores de base de datos ya que permitirá la elección adecuada de qué estructura de datos es la más apropiada para contener datos, según las características de estos.

La presente tesis ayudará también a los investigadores de ciencias de la computación como referencia y fuente verificable de análisis de velocidad en el acceso a los datos en memoria en función al tipo de estructuras utilizadas.

1.3. Objetivos

1.3.1. Objetivo general

Determinar la velocidad de respuesta de los algoritmos de búsqueda de datos contenidos en estructuras estáticas y dinámicas.

1.3.2. Objetivos específicos

- ✓ Determinar los tiempos de respuesta en la búsqueda de datos localizados en estructuras estática y en estructuras dinámicas.
- ✓ Determinar el número de comparaciones para localizar un dato contenido en estructuras dinámicas y estáticas.

1.4. Hipótesis

1.4.1. Hipótesis general

H_o: La velocidad de respuesta en las operaciones de búsqueda de datos contenidos en estructuras dinámicas será mayor que la velocidad de respuesta en la búsqueda de datos contenidos en estructuras de datos estáticas.

H_a: La velocidad de respuesta en las operaciones de búsqueda de datos contenidos en estructuras dinámicas será menor que la velocidad de respuesta en la búsqueda de datos contenidos en estructuras de datos estáticas.

1.4.2. Hipótesis específica

Hipótesis específica 1:

H_o. El tiempo de respuesta en la búsqueda de datos localizados en estructuras dinámicas no lineales tipo Árbol es menor que el tiempo de respuesta en la búsqueda de datos localizados en estructuras estáticas tipo Array.

H_a. El tiempo de respuesta en la búsqueda de datos localizados en estructuras dinámicas no lineales tipo Árbol es mayor que el tiempo de respuesta en la búsqueda de datos localizados en estructuras estáticas tipo Array.

Hipótesis específica 2:

H_o. El número de comparaciones necesarias para localizar un dato contenido en una estructura dinámica no lineal tipo Árbol es menor que el número de comparaciones necesarias para localizar un dato contenido en una estructura estática tipo Array.

H_a. El número de comparaciones necesarias para localizar un dato contenido en una estructura dinámica no lineal tipo Árbol es mayor que el número de comparaciones necesarias para localizar un dato contenido en una estructura estática tipo Array.

1.5. Variables

- ✓ Velocidad de respuesta en la búsqueda de datos
- ✓ Estructuras de datos en memoria

1.5.1. Caracterización de las variables

Variable dependiente: Velocidad de respuesta de los algoritmos de búsqueda de datos.

Indicadores:

- ✓ Tiempo de ejecución del algoritmo
- ✓ Número de comparaciones de datos

Variable independiente

Estructuras de datos estáticas y dinámicas.

Indicadores:

- ✓ Espacio de memoria reservado para almacenamiento de datos
- ✓ Espacio real de memoria utilizado para almacenar datos

1.5.2. Operacionalización de las variables

La velocidad de respuesta de los algoritmos de búsqueda de datos se medirá utilizando funciones que capturan los tiempos de ejecución entre diferentes puntos del código fuente en los lenguajes de programación de alto nivel. En nuestro caso utilizaremos un compilador del lenguaje C++ y preferentemente en modo consola y no en modo visual para evitar el consumo de recursos computacionales que puedan distorsionar en objetivo fundamental de nuestra investigación que consiste en medir básicamente los tiempos que tardan los procesos computacionales para diferentes estructuras de datos.

El instrumento a utilizar serán reportes documentales generados por el ordenador, previa codificación en el mismo lenguaje C++. En el caso de las estructuras de datos estáticas y dinámicas mediremos el espacio de memoria utilizado por cada estructura. La medición se efectuará para el mismo volumen de datos y con las mismas características o tipos de datos. Esto para que las condiciones de medición sean homogéneas.

CAPÍTULO II

MARCO TEÓRICO

2.1. Antecedentes del estudio

Para Peña y Suarez (2005) en su tesis titulada “Utilización de información histórica para decisiones empresariales”, los atributos de tipo de texto que describen cosas son organizados en dimensiones. Es necesario establecer un criterio puramente de diseño y basado en los requerimientos del negocio para establecer los atributos que se incluyen como dimensiones y los que se pueden descartar al realizar el almacenamiento de datos.

Los atributos dimensionales, servirán como fuente para las restricciones y encabezados de filas en los reportes. Todos los atributos dimensionales son igualmente candidatos a ser encabezados en filas en los reportes.

El proceso de búsqueda en estructuras estáticas funciona bien para bodegas de datos que no manejan grandes cantidades de registros, pues al ser grande el tamaño de datos es preferible mantener una tabla de búsqueda que registre la llave primaria en la fuente y la llave subrogada

en esta dimensión. De esta manera se mejora el rendimiento al poblar las tablas de hechos.

Murillo Morera, J. (2012) en el Artículo Científico titulado: Comparación entre algoritmos recursivos e iterativos y su medición en términos de eficiencia concluye que en comparaciones simples entre algoritmos recursivos e iterativos, para determinar el grado de eficiencia de un problema en particular se efectuaron pruebas de comparación y análisis utilizando tres ejemplos en ambos tipos de algoritmos, a los cuales se les aplicaron los criterios de análisis de algoritmos obtuvieron mayor tiempo de respuesta en el caso de los procesos recursivos pero menor tamaño de código. El autor concluye que los procesos recursivos sólo se deben emplear en casos que no se pueda resolver por el método iterativo.

Amalia Duch, (2007) en su obra Algoritmos, señala respecto al tiempo y costo computacional de los algoritmos que La característica básica que debe tener un algoritmo es que sea correcto, es decir, que produzca el resultado deseado en tiempo finito. Adicionalmente puede interesarnos que sea claro, que esté bien estructurado, que sea fácil de usar, que sea fácil de implementar y que sea eficiente.

Cairó, O. (2007) en su Libro Estructura de datos señala que, el tiempo de respuesta en el ordenador, de los algoritmos recursivos es menor porque las pilas de recursión consumen abundante memoria del ordenador, no así los algoritmos iterativos que en identificadores o variables simples van actualizando los valores de proceso computacional. Así mismo sugiere que sólo se utilice la recursión cuando el diseño de la solución recursiva sea demasiada compleja.

2.2. Bases Teóricas

Los datos son costosos. Deben ser manejados de tal manera que sean correctos y estén disponibles para producir información (Loomis, 1999, Estructura de Datos y Organización de Archivos, p. 3).

Los costos por usar una lista ligada (estructura dinámica), en lugar de una representación secuencial (estructuras estáticas) son dos: Primero, el espacio requerido para una representación ligada requiere una cantidad extra para el campo del apuntador. Segundo, con frecuencia la búsqueda de un nodo en una representación ligada será más larga con respecto a la búsqueda en un arreglo que aloje a la lista. Recordemos que podemos calcular la localidad del inicio del n-ésimo elemento en un arreglo. Para encontrar el n-ésimo nodo en una lista ligada (suponiendo que no utilizamos punteros auxiliares), la cadena de los primeros N-1 nodos debe ser visitada, ya que el n-ésimo nodo no puede ser accesado

directamente. (Loomis, 1999, Estructura de Datos y Organización de Archivos, p. 3).

Muchos algoritmos requieren una representación apropiada de los datos para lograr ser eficientes. Esta representación junto con las operaciones permitidas se llama estructura de datos. Típicamente todas las estructuras e datos permiten inserciones arbitrarias. Las estructuras de datos varían en cómo permiten el acceso a miembros del grupo. Algunas permiten tanto accesos como operaciones de borrado arbitrarios. Otras imponen restricciones, tales como permitir el acceso sólo al elemento más recientemente insertado. (Weiss, 2004, Estructuras de Datos en Java, p. 137).

Un algoritmo de búsqueda es un algoritmo que acepta un argumento a y trata de encontrar un registro cuya llave es a. El algoritmo puede retornar el registro completo o con frecuencia retorna un puntero a ese registro. Es posible que la búsqueda por algún argumento en particular en una tabla no tenga éxito; es decir, que no exista registro en la tabla con ese argumento como llave. En este caso, el algoritmo debe retornar un registro “nulo” o un “puntero nulo”. Frecuentemente si la búsqueda no tiene éxito, puede ser mejor adicionar un nuevo registro con el argumento como llave. Un algoritmo que hace esto se llama algoritmo de búsqueda e inserción.

La forma como están organizados pueden ser arreglos de registros, una lista encadenada, un Árbol, o inclusive un grafo. Debido a que diferentes técnicas de búsqueda pueden ser apropiadas para diferentes organizaciones. La tabla generalmente está diseñada para alguna técnica de búsqueda específica. La tabla puede estar contenida en forma completa en memoria, completamente en un almacenamiento auxiliar, o puede estar partida entre los dos. En este caso, se requieren diferentes técnicas de búsqueda de acuerdo a las diferentes condiciones que se asuman. Aquellos tipos de búsqueda en las cuales la tabla está completamente contenida en la memoria, son llamados búsquedas internas, mientras que aquellas en las cuales casi toda la tabla es mantenida en almacenamiento auxiliar son denominadas búsquedas externas. (Tenenbaum, 2004, Estructura de Datos en C, p.437).

La forma más simple del método de búsqueda es la búsqueda secuencial. Este método es aplicable a una tabla que está organizada ya sea como un arreglo o como una lista encadenada. Asumamos que k es un arreglo de n llaves y r un arreglo de registros tal que k(i) es la llave de r(i). Asumamos también que i es el argumento de búsqueda. El objetivo es asignar la variable (o función identificadora) search() al entero más pequeño i tal que k(i) = key si este i existe, y en caso contrario será igual a cero.

El almacenar una tabla como una lista encadenada tiene la ventaja que el tamaño de la tabla puede ser aumentada dinámicamente de acuerdo a los requerimientos. Asuma que la tabla está organizada como una lista encadenada lineal apuntada por table y encadenada por un campo puntero denominado next (Tenenbaum, 2004, Estructura de Datos en C, p.439).

Como la búsqueda es una actividad tan común en computación, es conveniente encontrar un método eficiente para ejecutarla. Quizás el método de búsqueda menos refinado es el de búsqueda lineal o secuencial, en el cual se examina cada uno de los elementos del arreglo y se le compara, cada vez, con el que se busca hasta que ocurre una coincidencia. Si la lista está desordenada y construida al azar, la búsqueda lineal puede ser la única vía de encontrar algo en ella (a no ser, por supuesto, que se ordene primero). Sin embargo, no debería usarse éste para buscar un nombre en un directorio telefónico. En su lugar, se abre el libro en una página cualquiera y se examinan los nombres que aparecen en ella. Como están ordenados en forma alfabética, dicho examen determinará si debe continuarse la búsqueda en la primera o segunda mitad del libro.

Si el arreglo sólo contiene un elemento, el problema es trivial. En otro caso, compárese el elemento que se busca con el elemento central

del arreglo. Si son iguales, se ha concluido con éxito la búsqueda. Si el elemento central es mayor, se repite el proceso de búsqueda en la primera mitad del arreglo (dado que si el elemento buscado aparece en algún lugar debe hacerlo en la primera mitad): en caso contrario, se repite el proceso con la segunda mitad. Obsérvese que cada vez que se realiza una comparación, se divide en dos el número de elementos que aún deben buscarse. Para arreglos grandes éste método es superior al de búsqueda secuencial en el cual cada comparación reduce sólo en 1 el número de elementos que aún deben examinarse. Este método se llama búsqueda binaria, debido a que el arreglo donde se realiza la búsqueda se divide en dos partes iguales. (Tenenbaum, 2004, Estructura de Datos en C, p.110).

El método de búsqueda más eficiente en una tabla secuencial sin usar índices o tablas auxiliares es el de búsqueda binaria. Básicamente se compara el argumento con la llave del elemento medio de la tabla. Si son iguales, la búsqueda termina con éxito; en caso contrario, se busca de manera similar en la mitad superior o inferior de la tabla. (Tenenbaum, 2004, Estructura de Datos en C, p.398).

Hay una técnica para perfeccionar la eficiencia de la búsqueda en un archivo ordenado, pero involucra un incremento en la cantidad de espacio requerido. Este método se llama método de búsqueda secuencial

indexada. Se aparta una tabla auxiliar, llamada index además del propio archivo ordenado. Cada elemento en el index consta de una llave kindex. Los elementos en el índice tanto como los elementos en el archivo, tienen que estar ordenados de acuerdo a las llaves. Si el índice es un octavo del tamaño del archivo, cada octavo registro del archivo tiene que estar representado en el índice. (Tenenbaum, 2004, Estructura de Datos en C, p.396).

Otra técnica para buscar en un arreglo ordenado es la llamada búsqueda por interpolación. Si las llaves están distribuidas de manera uniforme entre k(0) y k(n-1), el método puede ser aun más eficiente que la búsqueda binaria. (Tenenbaum, 2004, Estructura de Datos en C, p.401).

Un Árbol binario de búsqueda, puede usarse como un Árbol de búsqueda binaria. Usando notación de Árbol Binario, el algoritmo para la búsqueda de la llave key en un Árbol de este tipo es como sigue (suponemos que cada nodo contiene cuatro campos: k, que guarda el valor de la llave del registro, r, que guarda el propio registro y left y right que son apuntadores a los sub Árboles).

La eficiencia del proceso de búsqueda puede perfeccionarse usando un centinela como en la búsqueda secuencial. Un nodo centinela con un apuntador externo por separado apuntándole, permanece asignado con el Árbol. Todos los apuntadores al Árbol left o right que no apunten a otro

nodo del Árbol apuntan ahora a ese centinela en lugar de ser iguales al apuntador nulo. Cuando se ejecuta una búsqueda, se inserta primero la llave argumento en el nodo centinela, garantizando así que será encontrado en el Árbol. (Tenenbaum, 2004, Estructura de Datos en C, p.406).

El Árbol Binario de Búsqueda es una estructura sobre la cual se pueden realizar eficientemente las operaciones de búsqueda, inserción y eliminación. Comparando esta estructura con otras, puede observarse ciertas ventajas. Por ejemplo, en un arreglo es posible localizar datos eficientemente si los mismos se encuentran ordenados, pero las operaciones de inserción y eliminación resultan costosas. En las listas, las operaciones de inserción y eliminación se pueden llevar a cabo con facilidad, sin embargo la búsqueda es una operación bastante costosa que incluso nos puede llevar a recorrer todos los elementos de ella para localizar uno en particular. (Cairó, 2007, Estructura de Datos, p.208).

La búsqueda secuencial consiste en revisar elemento por elemento hasta encontrar el dato buscado, o hasta llegar al final de la lista de datos disponibles.

Cuando se habla de búsqueda en arreglos debe distinguirse entre arreglos desordenados y arreglos ordenados. La búsqueda secuencial en arreglos desordenados consiste, básicamente, en recorrer el arreglo de

izquierda a derecha hasta que se encuentre el elemento buscado o se termine el arreglo, lo que ocurra primero. Normalmente cuando una función de búsqueda concluye con éxito, interesa conocer en qué posición en qué posición fue hallado el elemento buscado. Esta idea puede generalizarse para todos los métodos de búsqueda.

La búsqueda secuencial en arreglos ordenados es similar al caso anterior. Sin embargo, el orden que existe entre los elementos del arreglo permite incluir una nueva condición que hace más eficiente al proceso.

El método de búsqueda secuencial también puede aplicarse a listas ligadas. Consiste en recorrer la lista, nodo por nodo, deteniéndose cuando haya encontrado el valor buscado, o cuando haya revisado todos los nodos de la lista. El orden en el cual puede recorrerse la lista depende de las características de la misma. Es decir, si es simple o doblemente ligada, y además si es circular. (Cairó, 2007, Estructura de Datos, pp.352-353).

La búsqueda binaria consiste en dividir el intervalo de búsqueda en dos partes, comparando el elemento buscado con el central. En caso de no ser iguales se redefinen los extremos del intervalo (según el elemento central sea mayor o menor que el buscado) disminuyendo el espacio de búsqueda. El proceso concluye cuando el elemento es encontrado, o bien cuando el intervalo de búsqueda se anula.

Este método sólo funciona para arreglos ordenados. Con cada iteración del método el espacio de búsqueda se reduce a la mitad, por lo tanto el número de comparaciones a realizar disminuye notablemente. Esta disminución resulta significativa cuanto más grande sea el tamaño del arreglo. (Cairó, 2007, Estructura de Datos, pp.354-355).

Dada la naturaleza de los dispositivos de almacenamiento secundario, como los discos, el tiempo para encontrar un bloque y leerlo a la memoria principal es muy grande, comparado con el tiempo que lleva procesar el dato. Por ejemplo, supóngase que se tiene un bloque de 1000 enteros dentro de un disco que gira a 1000rpm. El tiempo que lleva colocar el cabezal sobre la pista que contenga este bloque (tiempo de búsqueda), mas el tiempo consumido en esperar que el bloque quede bajo la cabeza (tiempo de latencia), puede ser de 100 milisegundos en promedio. El proceso de escritura de un bloque en un lugar particular dentro del almacenamiento secundario lleva una cantidad similar de tiempo. Sin embargo, la máquina puede efectuar 100 000 instrucciones en esos 100 milisegundos. Este tiempo es más que suficiente para hacer un procesamiento simple a los mil enteros una vez que están en la memoria principal, cómo la suma de ellos o la ubicación del máximo; incluso puede ser suficiente para clasificación rápida de los enteros.

Cuando se evalúa el tiempo de ejecución de los algoritmos que operan sobre datos almacenados en forma de archivos, es de primordial importancia considerar el número de veces que se lee un bloque a la memoria principal o se escribe un bloque en la memoria secundaria; esa operación se denomina acceso a bloques. Se supone que el tamaño de los bloques lo fija el sistema operativo, así que no se puede hacer que un algoritmo se ejecute con mayor rapidez incrementando el tamaño del bloque, para reducir el número de accesos a los bloques. Como consecuencia, lo que se debe considerar para los algoritmos que emplean almacenamiento externo será el número de accesos a los bloques. (Aho, 1998, Estructura de Datos y Algoritmos, pp.347-348).

2.3. Clasificación de las estructuras de datos

Según Osvaldo Cairo (2002, p. 169), las estructuras de datos se pueden clasificar de dos maneras. De acuerdo a la posibilidad de modificación de las estructuras en tiempo de ejecución y de acuerdo al consumo de memoria en el ordenador, las estructuras de datos se clasifican como se muestra en la Tabla 1.

Tabla 1.

Estructuras estáticas y dinámicas

ESTRUCTURAS ESTÁTICAS	ESTRUCTURAS DINÁMICAS
Arreglos Registros Conjuntos	Listas Árboles

Fuente: Adaptado de “Estructura de datos” Osvaldo Cairo O, 2002

Otra clasificación se efectúa de acuerdo a la forma que enlazan los datos en posiciones de memorias.

Tabla 2.

Estructuras lineales y no lineales

ESTRUCTURAS LINEALES	ESTRUCTURAS NO-LINEALES
Arreglos Registros Pilas Colas Listas	Árboles Grafos

Fuente: Adaptado de “Estructura de datos” por Osvaldo Cairo O, 2002

Como se puede ver en la Tabla 2, se consideran como estructuras de datos no-lineales a los Árboles y a los grafos, sin embargo la estructura

tipo Árbol es en sí un tipo especial de grafo (dirigido, unidireccional, convexo y sin ciclos) (Sisa & Velez Muñoz, 2002, pág. 198).

Las estructuras lineales, son aquellas que al representarse en el hardware del ordenador, lo hacen situando sus elementos de forma contigua en relación de 1 a 1; esto quiere decir que cada elemento de la estructura sólo puede ir enlazado al elemento siguiente o anterior.

Las estructuras no-lineales se caracterizan por no existir una relación de sus elementos es decir que un elemento puede estar enlazado con uno o más elementos.

2.3.1. Árboles

Los Árboles son las estructuras de datos dinámicas y no-lineales más importantes en la computación. Son dinámicas, puesto que la estructura Árbol puede cambiar en tiempo de ejecución. Y son no-lineales, puesto que a cada elemento del Árbol pueden seguirle uno o varios elementos.

Algunas definiciones de esta estructura de datos son:

"Intuitivamente el concepto de Árbol implica una estructura en la que los datos se organizan de modo que los elementos de información están relacionados entre sí a través de ramas. Un Árbol consta de un conjunto finito de elementos, denominados «nodos» y un

conjunto finito de líneas dirigidas, denominadas «ramas» que conectan los nodos" (Joyanes Aguilar, 2002, pág. 499).

"La organización de los datos en una estructura no lineal, jerárquica o de niveles, es una opción para representar estructura de datos, las más conocidas y usadas en la computación se denominan Árboles. Su característica principal es que mantienen una relación de uno a muchos (1:n) entre sus elementos" (Martinez & Quiroga, 2002, pág. 116).

"Formalmente se define un Árbol de tipo T como una estructura homogénea que es la concatenación de un elemento de tipo T junto con un número finito de Árboles disjuntos, llamados sub Árboles. Una forma particular de Árbol puede ser la estructura vacía" (Cairó Battistutti & Guardati Buemo, 2002, pág. 170).

De las definiciones anteriores podemos concluir que un Árbol es un tipo de estructura de datos no lineal, jerárquica y homogénea aplicada sobre una conjunto de objetos llamados nodos (uno de los cuales es conocido como raíz). Debido a la estructura jerárquica de los nodos, se establece una relación de parentesco entre los nodos, dando lugar a términos como padre, hijo, hermano, antecesor, sucesor, ancestro, etc.

Los Árboles son estructuras muy comunes, se pueden encontrar varios ejemplos en la vida cotidiana. La mayoría de las personas, por ejemplo, denominan Árbol genealógico a su linaje (Bowman, 1999). Como ejemplo se tiene un organigrama de una institución en la Figura 1.

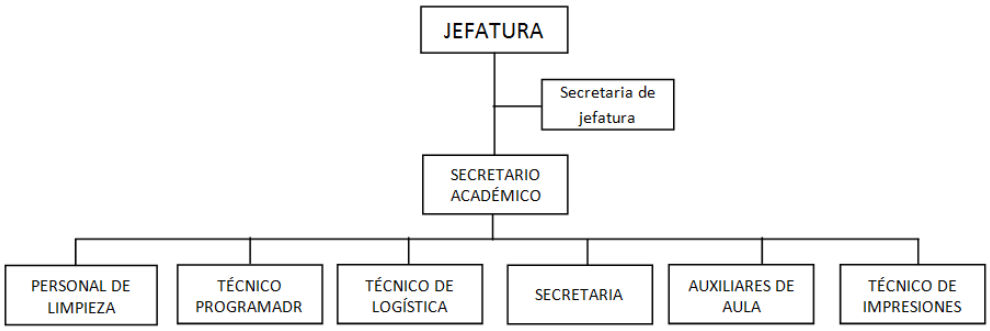


Figura 1. Organigrama del CEPU-UNJBG

Fuente. Centro Pre Universitario - UNJBG

Se observa que por comodidad se dibuja la raíz en la parte superior del diagrama y las hojas en la parte inferior.

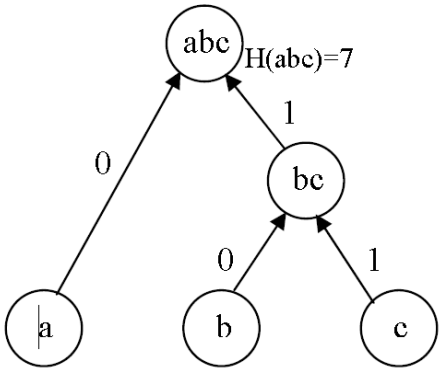


Figura 2. Estructura tipo Árbol

Fuente: Elaboración propia

La estructura tipo Árbol se expresa generalmente de manera gráfica como se muestra en la Figura 2. Este tipo de estructura tiene una gran variedad de aplicaciones.

En Ciencias de la Computación, los Árboles se usan exhaustivamente como una estructura eficiente para realizar búsquedas en lista dinámicas grandes y para aplicaciones diversas como los sistemas de inteligencia artificial y los algoritmos de codificación. (Forouzan & Chung Fegan, 2003, pág. 334).

Definición recursiva de Árbol

Los estructuras tipo Árbol tienen una estructura recursiva porque es la forma en que se representa más apropiadamente y además es una característica inherente a los mismos.

“Un Árbol es un conjunto finito de uno o más nodos tal que existe un nodo especial llamado raíz y los demás nodos (excepto la raíz) forman conjuntos disjuntos que a su vez son Árboles” (Knuth, 1973, pág. 23).

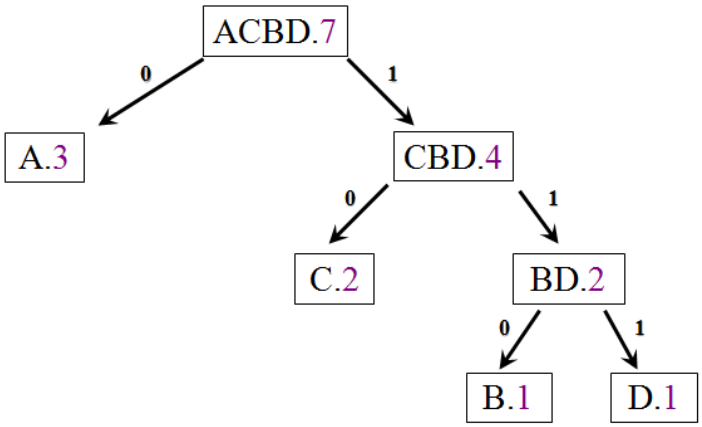


Figura 3. Naturaleza recursiva de un Árbol.

Fuente: Elaboración propia

Representaciones de un Árbol

Gráficamente, una estructura Árbol puede representarse de numerosas maneras. En la siguiente Figura 4 se muestran cuatro de ellas.

En la figura 4.a se muestra la representación gráfica que es sin lugar a dudas la más utilizada y la que mejor pone de manifiesto las relaciones entre los nodos; en la figura 4.b se representa por medio de diagramas de Venn; en la figura 4.c por medio de la notación indentada; y en la figura 4.d se muestra por representación de paréntesis anidados que ofrece por su parte la ventaja de ser más la representación más compacta.

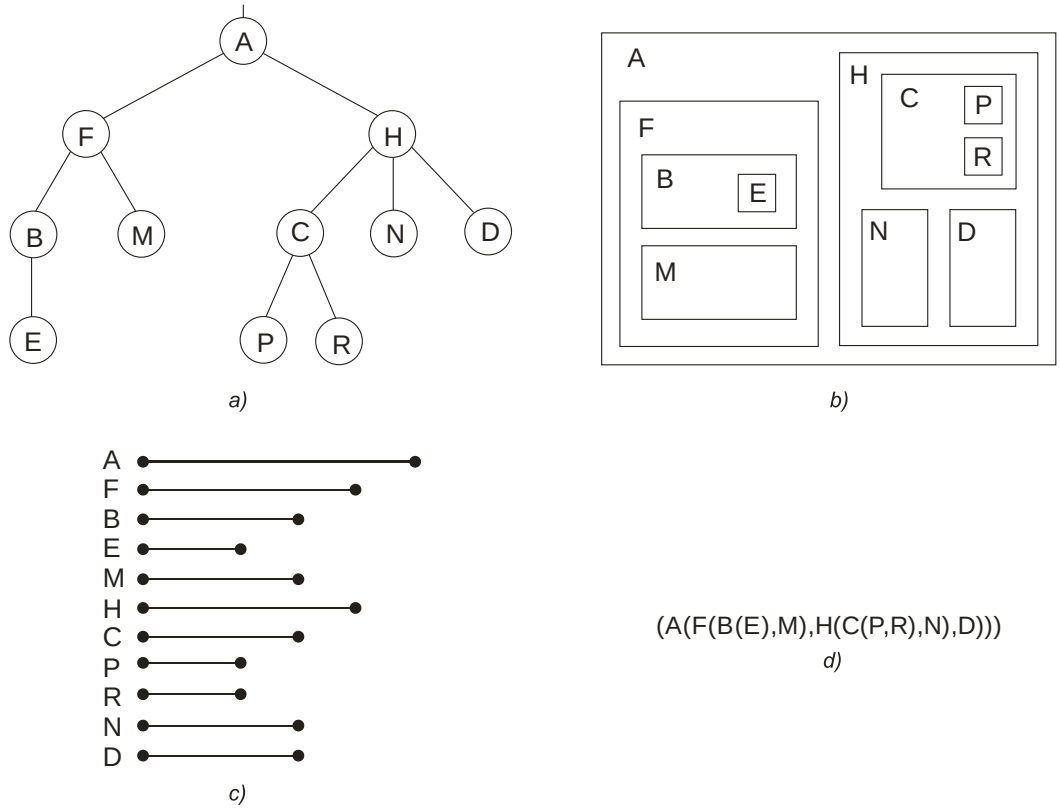


Figura 4. Representación de la estructura tipo Árbol.
a) Grafo. b) Diagrama de Venn. c) Notación indentada. d) Anidación de paréntesis.
Adaptado de “Estructura de datos” por Osvaldo Cairo, 2002, p. 171.

Terminología usada en Árboles

Los Árboles se basan en el concepto de nodo. Hay otras estructuras de datos que también hacen uso de este concepto como se ilustra en la Figura 5. En general, los nodos pueden definirse de la siguiente forma:

Se denomina nodo a cualquier tipo cuyos elementos son registros formados por un campo “Datos” y un número dado de apuntadores o enlaces. (Hernández, Lázaro, Dormido, & Ros, 2000, pág. 194).

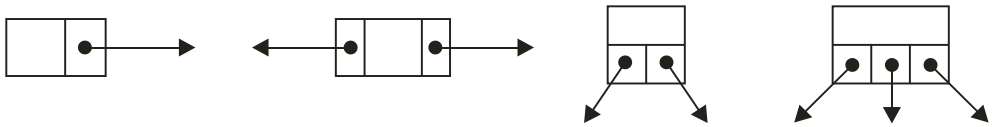


Figura 5. Concepto de nodo.
Se muestran, los nodos correspondientes a una lista enlazada, un Árbol Binario y un Árbol Ternario, en ese orden.

Adaptado de “Estructura de datos y algoritmos” por Roberto Hernández, 2001, p. 194.

En el estudio de los Árboles se hace uso de la siguiente terminología, que se ilustra en la Figura 6.

Raíz: Es un único nodo en donde nace toda la estructura, no tiene predecesor, pero si puede tener sucesores.

Nodos hoja: Nodos que no tienen sucesor, se denominan también nodos terminales.

Nodos interiores: Nodos que no son ni raíz, ni nodos hoja.

Sub-Árboles: Si se aplica la definición recursiva de Árbol, un sub-Árbol es cada uno de los Árboles que salen de un nodo. También se denominan ramas.

Nivel: Los niveles se establecen de la siguiente manera: A la raíz se le asigna el nivel uno, los inmediatos sucesores de un nodo tienen un nivel más que éste, y así sucesivamente, lo que implica que todos los inmediatos sucesores de un nodo tienen el mismo nivel. Al aplicar este concepto a la figura 4, se obtiene la Tabla 3:

Tabla 3.

Niveles del Árbol de la figura 4

NIVEL	**NODOS**
1	A
2	F H
3	B M C N D
4	E P R

Niveles en el Árbol. Se debe tener en cuenta que algunos autores asignan a la raíz el nivel cero.

Adaptado de “Estructura de datos y algoritmos” por Jaime Sisa, 2002, p. 201.

Altura del Árbol: La altura de un Árbol es el máximo número de niveles de todos los nodos del Árbol.

Grado: Es el número de sub-Árboles que salen de un nodo. Según el grado, los Árboles se denominan binarios (como máximo dos hijos), ternarios (como máximo tres hijos), y así sucesivamente. Los Árboles Binarios son de especial interés ya que cada nodo tiene dos descendientes, denominados sub Árbol izquierdo y sub Árbol derecho.

Todos los términos mostrados anteriormente, se ven reflejados en la Figura 6, se puede observar además la relación entre dos nodos x e y cualquiera, a la que llamaremos relación padre-hijo.

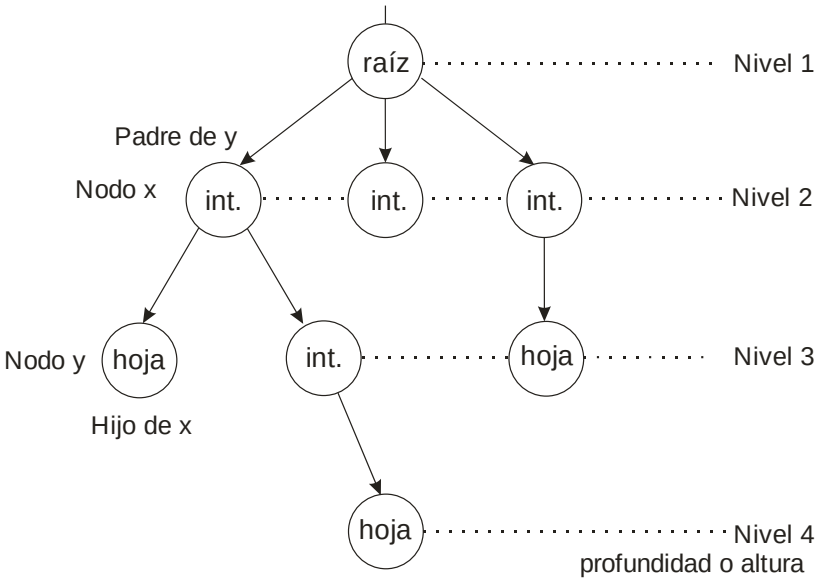


Figura 6. Terminología usada para los Árboles

Adaptado de “Estructura de datos y algoritmos” por Roberto Hernández, 2001, p. 196.

Forma de clasificar los Árboles

Los Árboles se pueden clasificar de diferentes maneras y por diferentes conceptos, a continuación se menciona la clasificación de Árboles, según Jaime Sisa (2002, p. 201), más usadas en informática:

ÁRBOLES HOMOGÉNEOS

Un Árbol homogéneo es aquel en el que todos sus nodos tienen la misma conformación, es decir, cada uno de los nodos que conforman el Árbol tiene los mismos campos de información y de apuntadores.

ÁRBOLES N-AREOS

De acuerdo con el número de sub Árboles que salen de los nodos, los Árboles se denominan binarios, si el número máximo de sub Árboles que sale de cualquier nodo es dos. Árboles Ternarios, si el número máximo de sub Árboles es tres. Árboles Cuaternarios, si el número máximo es cuatro. En general, se puede hablar de Árboles n–arios.

ÁRBOLES COMPLETOS

Un Árbol completo se caracteriza porque todos sus nodos hoja tienen la misma altura. Lo que implica que todos los enlaces de los

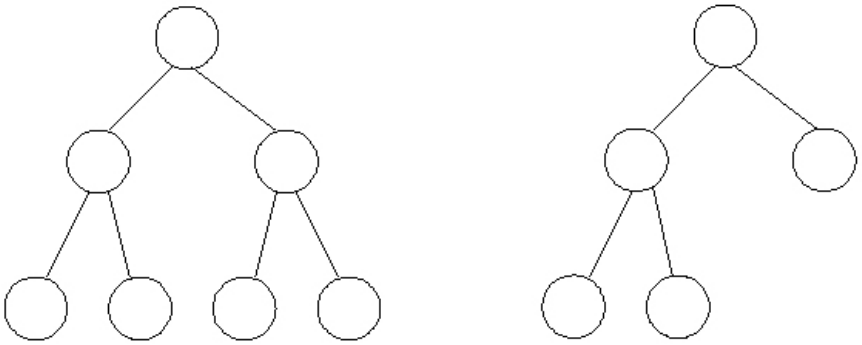
nodos interiores son diferentes de nulo. Normalmente estos Árboles son homogéneos.

ÁRBOLES ORDENADOS

Un Árbol ordenado se caracteriza porque la posición relativa de los sub Árboles es fija, es decir, no se pueden intercambiar. Esta denominación se usa normalmente para Árboles homogéneos.

ÁRBOLES BALANCEADOS

Un Árbol balanceado es aquel que además de ser homogéneo, se caracteriza porque intenta mantener su altura, o el número de niveles de nodos bajo la raíz, tan pequeños como sea posible en todo momento. Esto es importante, ya que muchas operaciones en un Árbol de Búsqueda binaria tardan un tiempo proporcional a la altura del Árbol, y los Árboles Binarios de Búsqueda ordinarios pueden tomar alturas muy grandes en situaciones normales, como cuando las claves son insertadas en orden. Mantener baja la altura se consigue habitualmente realizando transformaciones en el Árbol, como la rotación de Árboles, en momentos clave. En la Figura 7 se muestra unos ejemplos de Árboles balanceados.



a) b)

Figura 7. Árboles balanceados

a) Árbol Binario estrictamente balanceado o completo. b) Árbol Binario no estrictamente balanceado.

Fuente: Elaboración propia

a. Árboles Binarios

Los Árboles Binarios son Árboles n-arios cuyo número máximo de sub Árboles que sale de cualquier nodo es dos.

Son las estructuras más utilizadas en computación para el tratamiento de los datos en la memoria principal. Los Árboles Binarios tienen múltiples aplicaciones. Se usan para tomar decisiones de dos opciones, para representar un árbol genealógico, para presentar un campeonato clasificatorio de tenis, fútbol, entre otros. En la Figura 8 y en la Figura 9 se ilustran ejemplos de estas representaciones usando Árboles Binarios.

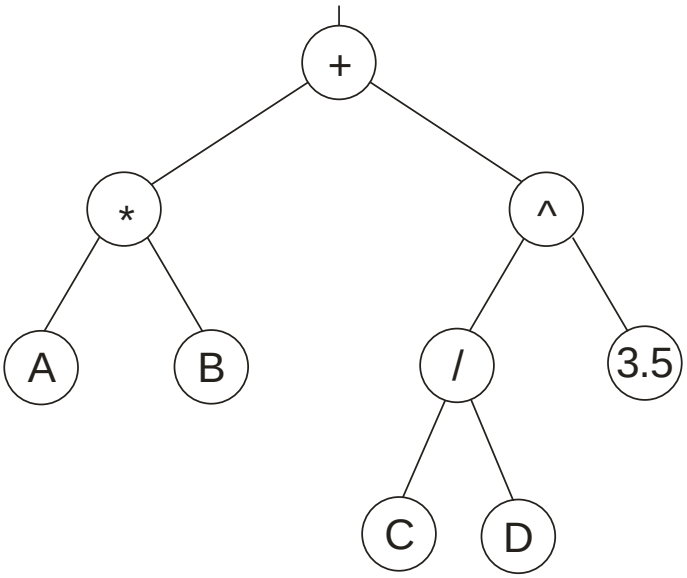


Figura 8. Representación de una expresión algebraica.

(A * B) + (C / D) ^3.5.

Fuente: Elaboración propia

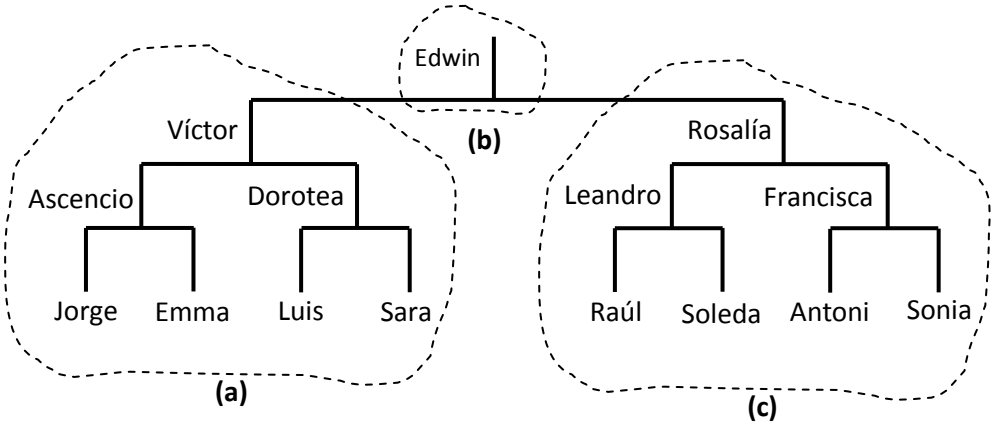


Figura 9. Ejemplo de Árbol genealógico.

a) Sub Árbol. b) Raíz. c) Sub Árbol 2.

Fuente: Elaboración propia

Definición de la clase Árbol Binario

En el apartado anterior se definió que es un Árbol n-ario. Un Árbol Binario es un Árbol cuyo número máximo de sub Árboles es 2 y por ende el Árbol es de grado 2. Estos Árboles son de especial interés puesto que representan una de las estructuras de datos más usadas en computación.

"Un Árbol Binario es un conjunto finito de elementos que puede estar vacío o contener un elemento denominado la raíz del Árbol y otros elementos divididos en dos subconjuntos separados, cada uno de los cuales es en sí un Árbol Binario. Estos dos subconjuntos son denominados sub Árbol izquierdo y subárbol derecho del árbol original. Cada elemento de un Árbol Binario se denomina un nodo del Árbol". (Tanenbaum & Augenstein, 1993, pág. 329).

Representación de un Árbol Binario en memoria

En las secciones anteriores se ha mostrado las aplicaciones de un Árbol Binario, así como su representación de manera gráfica y una definición formal de este. Pero, cuando necesitamos implementar un Árbol Binario en una computadora mediante un lenguaje de programación, tendremos que expresar el Árbol en forma que

comprenda el computador. Tradicionalmente se usan dos formas. (Cairó Battistutti & Guardati Buemo, 2002, pág. 180).

- Por medio de arreglos.
- Por medio enlaces tipo puntero.

"Para implementar un Árbol como un arreglo, un nodo se declara como un objeto con un campo de información y dos campos de «referencia». Estos campos de referencia contienen los índices de las celdas del arreglo en el cual se almacenan los hijos izquierdo y derecho si existen. Sin embargo esta implementación puede ser poco conveniente. Las localidades de los hijos deben conocerse para insertar un nodo nuevo, y estas localidades deben localizarse con forma secuencial. Después de eliminar un nodo del Árbol, tendría que eliminarse un hueco en el arreglo". (Drozdek, 2007, pág. 219)

La cita anterior nos deja en claro que implementar un Árbol Binario mediante un arreglo es poco conveniente, es por esto que en el presente trabajo se explicará la segunda forma de representar un Árbol Binario, es decir por medio enlaces tipo puntero (apuntadores).

Los nodos del Árbol Binario serán representados como registros, tal como se ilustra en la Figura 10, que contendrán como mínimo tres campos.

En un campo se almacenará la información del nodo. Los dos restantes se utilizarán para apuntar los sub Árboles izquierdo y derecho respectivamente del nodo en cuestión.

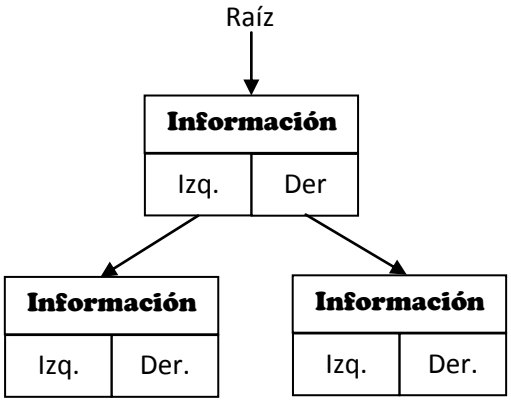


Figura 10. Representación de un Árbol Binario.

Fuente: Elaboración propia

Información: Campo donde se almacena la información de interés del nodo. En gran parte de la presente tesis se almacenará en este campo un valor simple numérico. Pero debemos tener en cuenta que en la práctica es común almacenar en este campo registros, arreglos e inclusive conjuntos.

Izq: Campo donde se almacena la dirección del sub Árbol izquierdo del Árbol.

Der: Campo donde se almacena la dirección del sub Árbol derecho del Árbol.

Tipos de recorrido en un Árbol Binario

Usualmente, los programas que trabajan con Árboles necesitan aplicar sistemáticamente un tratamiento a todos sus nodos en un orden dado por la forma del Árbol; es lo que se denomina visitar todos los nodos del Árbol.

"Se distinguen dos categorías básicas de recorrido: los que se basan en las relaciones padre-hijo de los nodos, que denominaremos recorridos en profundidad, y los que se basan en la distancia del nodo a la raíz conocidos como recorridos en anchura o por niveles." (Franch Gutiérrez, 1999, pág. 219).

En este trabajo de tesis mencionaremos los recorridos por profundidad por ser los más usados, y son los siguientes:

i. Recorrido en PREORDEN

   - Visitar la raíz
   - Recorrer el sub Árbol izquierdo
   - Recorrer el sub Árbol derecho

ii. Recorrido en INORDEN

   - Recorrer el sub Árbol izquierdo
   - Visitar la raíz

- Recorrer el sub Árbol derecho

iii. Recorrido en POSTORDEN

- Recorrer el sub Árbol izquierdo
- Recorrer el sub Árbol derecho
- Visitar la raíz.

Ha de notarse que estos tres tipo de recorrido, tienen naturaleza recursiva.

b. Árboles Binarios de Búsqueda (ABB)

En un arreglo (estructura lineal), la manera más eficiente de realizar una búsqueda es con el algoritmo de búsqueda binaria aplicado en una tabla de memoria estática. Sin embargo, esta estructura presenta la desventaja de no ser eficiente para la inserción y la eliminación de elementos. Por otro lado, una lista enlazada ordenada (estructura lineal) tiene mejor comportamiento en las inserciones y bajas de sus elementos, pero no en el algoritmo de búsqueda binaria.

Ante esta disyuntiva, contar con estructuras no lineales como los Árboles, representa una opción para conjuntar las características

positivas de estas estructuras lineales. La propuesta inmediata es el Árbol Binario de Búsqueda.

El Árbol Binario de Búsqueda es una estructura sobre la cual se pueden realizar eficientemente las operaciones de búsqueda, ya que las operaciones de inserción y eliminación se aseguran de que el Árbol Binario este optimizado para la este fin (Cairó Battistutti & Guardati Buemo, 2002, pág. 195).

Para lograr esta eficiencia, es necesario que se cumpla una condición importante que se resume en la siguiente cita:

“Los Árboles Binarios de Búsqueda son Árboles Binarios en los cuales se cumple que para cualquier registro X perteneciente al Árbol, todos los datos de los nodos a la izquierda de X son menores que el dato de X y todos los datos a la derecha de X son mayores que el dato de X.” (Flores Rueda, 2005, pág. 148).

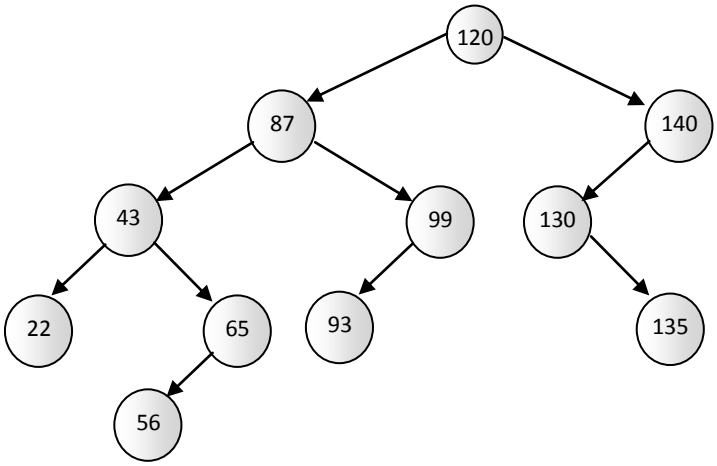


Figura 11. Árbol Binario de Búsqueda

Extraído de “Estructura de datos” por Osvaldo Cairo O, 2002, p. 196.

En la Figura 11 se presenta un Árbol Binario de Búsqueda. Si se realiza un recorrido inorden sobre un Árbol de Búsqueda obtendremos una clasificación de los nodos en forma ascendente. Los diferentes recorridos en el Árbol de la Figura 11 producen el siguiente resultado:

Preorden: 120 – 87 – 43 – 22 – 65 – 56 – 99 – 93 – 140 -130 – 135

Inorden: 22 – 43 – 56 – 65 – 87 – 93– 99 – 120 – 130 – 135 – 140

Postorden: 22 – 56 – 65 – 43 – 93 – 99 – 87 – 135 – 130 – 140 – 120

Operaciones en un Árbol Binario de Búsqueda

INSERCIÓN

Se efectúa al querer insertar un nuevo nodo en el Árbol; se debe cumplir la definición de Árbol Balanceado de Búsqueda y determinar la ubicación de los nodos en el Árbol: para cada nodo X, las claves de su sub Árbol izquierdo deben ser menores que la clave de nodo y las claves de su sub Árbol derecho deben ser mayores que la clave del nodo X. Los nodos se van insertando en el orden en que van llegando, así que el primero es el nodo raíz.

ELIMINACIÓN

Se basa en la búsqueda o consulta y permite eliminar cualquier nodo del Árbol. Se presentan los siguientes casos:

El caso más sencillo es aquel en que el nodo por eliminar es un nodo terminal u hoja. Sencillamente, para eliminarlo se desconecta del nodo padre.

Un segundo caso se presenta cuando el nodo por eliminar tiene sólo un sub Árbol; en este caso, el nodo que se elimina se remplaza por la raíz de su sub Árbol. Un tercer caso, que indiscutiblemente es el de mayor complejidad, se presenta cuando el nodo por eliminar tiene los dos sub Árboles; en este caso, el proceso

es el siguiente: se remplaza el nodo eliminado por el nodo de mayor valor de clave en su sub Árbol izquierdo, hay que notar que esto implica unos movimientos adicionales para conservar la estructura del Árbol y su información.

CONSULTA

Se usa para examinar la información correspondiente dado un dato. Para ello se aplica la norma que se utilizó para la inserción y se determina si se encuentra el elemento. Si es así, se retorna la información.

Árboles auto-balanceables

En las secciones anteriores se definió que los Árboles Binarios de Búsqueda son una estructura tipo Árbol especializada para la operación de búsqueda.

Sin embargo, hay ocasiones en que este tipo de estructura (ABB) pierde esa ventaja. Esto se da cuando el Árbol crece descontroladamente hacia un solo lado, de manera que su eficiencia en la búsqueda disminuye considerablemente.

En la Figura 12 se muestran dos casos en que un Árbol Binario de Búsqueda (ABB) crece hacia un solo lado del Árbol. Este caso se

da cuando se insertan datos de manera creciente o decreciente, lo que produce que el Árbol se degenere en una lista.

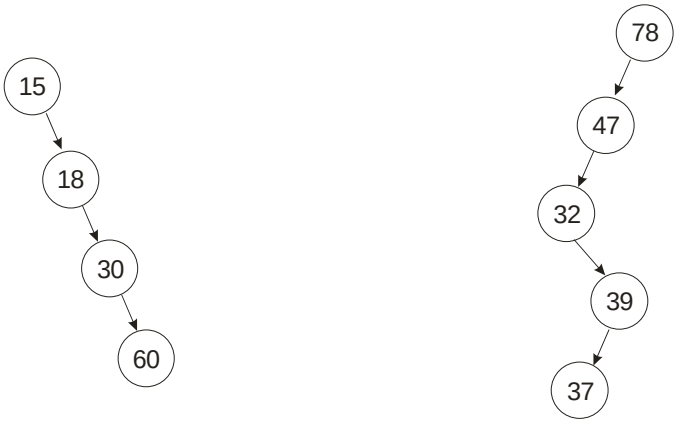
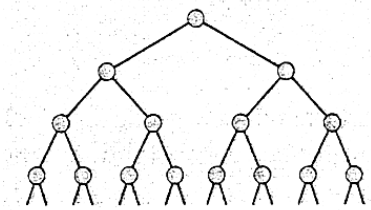


Figura 12. ABB con crecimiento descontrolado

Adaptado de “Estructura de datos” por Osvaldo Cairo O, 2002, p. 207.

Esto provocará que para encontrar un dato, se tendrá que recorrer los nodos de manera secuencial realizando un mayor número de comparaciones, perdiendo su eficiencia. Esta deficiencia se puede suplir si se controla que, mientras se inserten o eliminen datos, el Árbol se mantenga «balanceado».

Con el objeto de mejorar el rendimiento en la búsqueda surgen este tipo de estructuras. La idea central de éstos es la de realizar reacomodos, rotaciones o balances después de inserciones o eliminaciones de elementos.



Altura	*Nodos en un nivel*	*Nodos en todos los niveles*
1	$2^0 = 1$	$1 = 2^1 - 1$
2	$2^1 = 2$	$3 = 2^2 - 1$
3	$2^2 = 4$	$7 = 2^3 - 1$
4	$2^3 = 8$	$15 = 2^4 - 1$
⋮		
11	$2^{10} = 1\,024$	$2\,047 = 2^{11} - 1$
⋮		
14	$2^{13} = 8\,192$	$16\,383 = 2^{14} - 1$
⋮		
h	2^{h-1}	$n = 2^h - 1$
⋮		

Figura 13. Número máximo de nodos en Árboles Binarios.

Extraído de “Estructura de datos y algoritmos con Java” por Adam Drozdek, 2007, p. 250.

La Figura 13 muestra cuántos nodos pueden almacenarse en Árboles Binarios completos de diferentes alturas. Como cada nodo puede tener dos hijos, el número de nodos en un cierto nivel es el doble del número de padres que residen en el nivel anterior (excepto, por supuesto, la raíz). Por ejemplo, si 10000 elementos se almacenan en un Árbol perfectamente balanceado, entonces el Árbol tiene una altura de:

$$log_2(nh+1) = log_2(10\ 001) = 13.2891 = 14.\ (Drozdek,\ 2007,\ pág.\ 250)$$

Esto significa que si 10 000 elementos se almacenan en un Árbol «completamente» balanceado, entonces se revisarán a lo

mucho 14 nodos para encontrar un elemento en particular. Ésta es una diferencia importante si comparamos con las 10 000 pruebas que se realizarían en una arreglo o lista enlazada (en el peor caso).

Por lo tanto, un Árbol «completamente» balanceado minimiza la cantidad de pruebas que se transforma en menos tiempo de búsqueda. No obstante, el coste para mantener el Árbol completamente balanceado después de cada inserción es muy alto (Ziviani, 2007, pág. 156). Una forma de abordar este problema es hallar una solución intermedia que permita mantener el Árbol «casi balanceado» o auto-balanceado.

Un Árbol auto-balanceado es un Árbol que intenta mantener su altura, o el número de niveles de nodos bajo la raíz, tan pequeños como sea posible en todo momento y automáticamente. Esto es de gran ayuda, ya que las operaciones en un Árbol de Búsqueda Binaria tardan en la búsqueda un tiempo proporcional a la altura del Árbol, y los Árboles Binarios de Búsqueda ordinarios pueden tomar alturas muy grandes en situaciones normales, como cuando las claves son insertadas en orden creciente o decreciente.

El Árbol auto-balanceado más común y usado es el Árbol AVL que se detallará en la siguiente sección.

c. Árboles AVL

Un Árbol AVL, originalmente llamado Árbol admisible o “admisible tree” (Krishnamoorthy, 2008, pág. 530) es un Árbol Binario de Búsqueda auto-balanceado que trata de mantenerse lo más balanceado posible mientras se realizan operaciones de inserción y eliminación. Fue propuesto en 1962 por los matemáticos G. M. Adelson - Velskii y E. M. Landis en su artículo "An Algorithm for the organization of information" (Un algoritmo para la organización de la información).

Formalmente se define un Árbol AVL como un Árbol Binario de Búsqueda en el cual se debe cumplir la siguiente condición: “Para todo nodo del Árbol, la altura del sub Árbol derecho e izquierdo no debe diferir en más de una unidad”. (Caicedo Barrero, Wagner de García, & Méndez Parra, 2010, pág. 111).

La condición anterior es la que define al término «factor de equilibrio».

$$FE = H_{RD} – H_{RI}$$

Por lo tanto, el factor de equilibro de todos los nodos de un Árbol AVL puede ser -1, 0 o 1. (Gómez De Silva Garza & Ania Briseno, 2008, pág. 104)

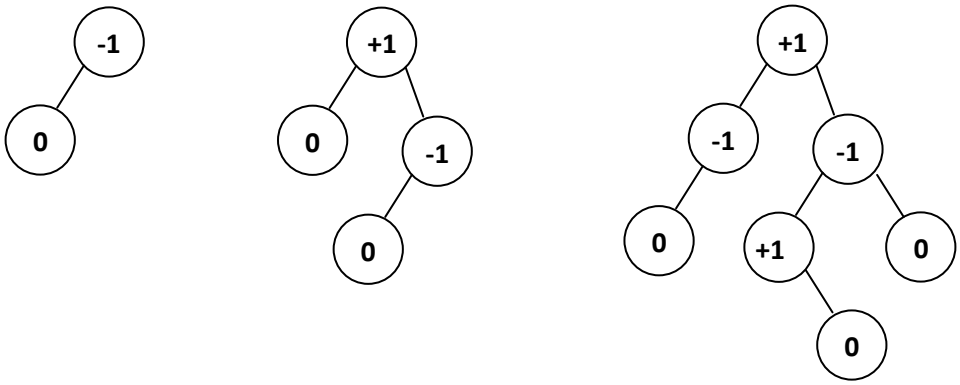


Figura 14. Ejemplos de Árboles AVL

Extraído de “Estructura de datos” por Osvaldo Cairo O, 2002, p. 208.

En la Figura 14, Cairo hace una representación gráfica de Árboles AVL. Dentro de cada nodo se observa su factor de equilibrio (FE). Los Árboles AVL facilitan el manejo del balanceo (Langsam, 1996) pues sólo se tiene en cuenta la diferencia de alturas de los sub Árboles; es decir que el balanceo de Árboles puede realizarse localmente si sólo afecta a una porción del Árbol cuando se requieren cambios después que un elemento se inserta o elimina en un Árbol.

OPERACIONES EN UN ÁRBOL AVL

INSERCIÓN

Al insertar un elemento en un Árbol AVL se distinguen los siguientes casos:

i. Las ramas izquierdas (RI) y derecha (RD) del Árbol tienen la misma altura ($H_{RI} = H_{RD}$), entonces:

   ✓ Si se inserta un elemento en RI entonces H_{RI} será mayor a H_{RD}.

   ✓ Si se inserta un elemento en RD entonces H_{RD} será mayor a H_{RI}.

   Note que en cualquiera de los dos casos mencionados (i.1 y i.2), no se viola el criterio de equilibrio del Árbol.

ii. Las ramas izquierda (RI) y derecha (RD) del Árbol tienen altura diferente ($HRI \neq HRD$):

   - Supóngase que $H_{RI} < H_{RD}$:

     - Si se inserta un elemento en RI entonces H_{RI} será igual a H_{RD}.

       (Las ramas tienen la misma altura, por lo que se mejora el equilibrio del Árbol)

     - Si se inserta un elemento en RD entonces se rompe el criterio de equilibrio del Árbol y es necesario reestructurarlo.

   - Supóngase que $H_{RI} > H_{RD}$:

- Si se inserta un elemento en RI, entonces se rompe el criterio de equilibrio del Árbol y es necesario reestructurarlo.
- Si se inserta el elemento RD, entonces H_{RD} será igual a H_{RI}.

  (Las ramas tienen la misma altura, por lo que se mejora el equilibrio del Árbol)

Ahora bien, para poder determinar si un Árbol está balanceado o no debe manejarse información relativa al equilibrio de cada nodo del Árbol. Esta información nos lo da el factor de equilibrio (FE). Si FE llegara a tomar los valores de -2 o 2, entonces debería reestructurarse el Árbol.

REESTRUCTURACIÓN

El proceso de inserción en un Árbol balanceado es sencillo pero con algunos detalles un poco complicados. Primero debe seguirse el camino de búsqueda del Árbol, hasta localizar el lugar donde hay que insertar el elemento. Luego se calcula su FE, que obviamente será 0, y regresamos por el camino de búsqueda calculando el FE de los distintos nodos. Si en alguno de los nodos se viola el criterio de equilibrio, entonces debe reestructurarse el Árbol.

El proceso termina al llegar a la raíz del Árbol, o cuando se realiza la restructuración del mismo, en cuyo caso no es necesario determinar el FE de los nodos restantes.

Reestructurar el Árbol significa rotar los nodos del mismo. En la Figura 15, se observa que la rotación puede ser simple o compuesta. El primer caso involucra dos nodos y el segundo caso afecta a 3. Si la rotación es simple puede realizarse por las ramas derechas (DD) o por las ramas izquierdas (II). Si la rotación es compuesta puede realizarse por las ramas derecha e izquierda (DI) o por las ramas izquierda y derecha (ID).

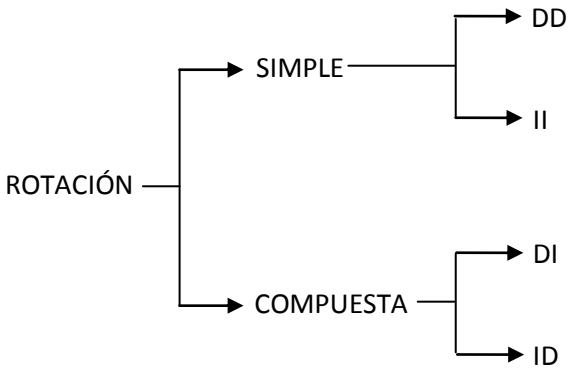


Figura 15. Tipos de rotaciones en un Árbol AVL.

Adaptado de “Estructura de datos” por Osvaldo Cairo O, 2002

Las rutinas de balanceo se pueden resumir en movimientos de apuntadores, y se ilustran en las figuras 16, 17, 18 y 19.

La nomenclatura que se usa en las imágenes para indicar la forma de balanceo es la siguiente: «raíz» es un apuntador que apunta a la raíz del sub Árbol por balancear, en todos los casos «raiz1» es un apuntador adicional que indica la dirección del nodo que está desbalanceado.

```
// ROTACIÓN DD
raiz->izq=raiz1->der;
raiz1->der=raiz;
raiz=raiz1;
```

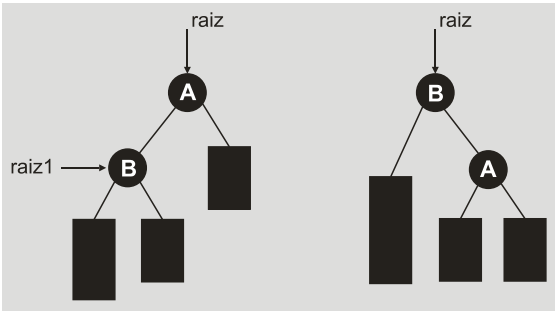


Figura 16. Rotación simple a la derecha (DD).

Adaptado de “Estructura de datos y algoritmos” por Jaime Sisa, 2002, p. 225.

```
// ROTACIÓN II
raiz->der=raiz1->izq;
raiz1->izq=raiz;
raiz=raiz1;
```

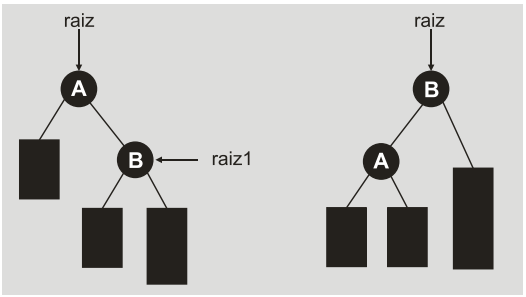


Figura 17. Rotación simple a la izquierda (II).

Adaptado de “Estructura de datos y algoritmos” por Jaime Sisa, 2002, p. 225.

```
// ROTACIÓN ID
raiz2=raiz1->der;
raiz->izq=raiz2->der;
raiz2->der=raiz;
raiz1->der=raiz2->izq;
raiz2->izq=raiz1;
raiz=raiz2;
```

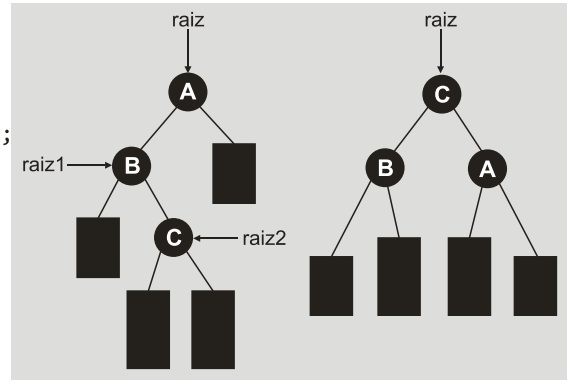


Figura 18. Rotación compuesta Izquierda Derecha (ID).

Adaptado de “Estructura de datos y algoritmos” por Jaime Sisa, 2002, p. 225.

```
// ROTACIÓN DI
raiz2=raiz1->izq;
raiz->der=raiz2->izq;
raiz2->izq=raiz;
raiz1->izq=raiz2->der;
raiz2->der=raiz1;
raiz=raiz2;
```

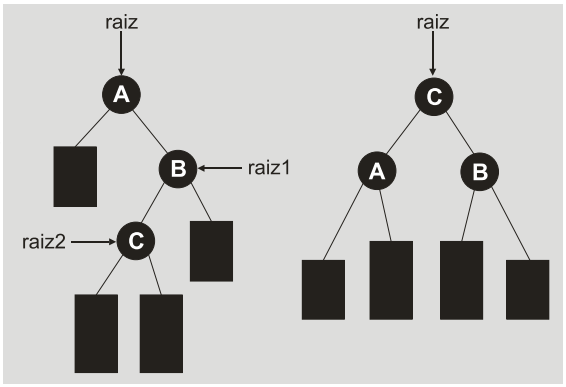


Figura 19. Rotación compuesta Derecha Izquierda (DI).

Adaptado de “Estructura de datos y algoritmos” por Jaime Sisa, 2002, p. 226.

ELIMINACIÓN

La operación de borrado en Árboles balanceados es un poco más compleja que la operación de inserción. Consiste en quitar un nodo del Árbol sin violar los principios que definen justamente un

Árbol balanceado. La definición nos indica que para todo nodo del Árbol, se debe cumplir que “la altura del sub Árbol izquierdo y la altura del sub Árbol derecho no debe diferir en más de una unidad”.

La complejidad en la operación de borrado, resulta ser cierta a pesar de que utiliza el mismo algoritmo de borrado (idéntico en la lógica, pero diferente en la implementación) de los Árboles Binarios de búsqueda y las mis operaciones de reacomodo que se utilizan en el algoritmo de inserción en Árboles balanceados.

Se debe recordar que en la operación de borrado en Árboles balanceados deben distinguirse los siguientes casos:

- Si el elemento a borrar es terminal u hoja, simplemente se suprime.
- Si el elemento a borrar tiene un solo descendiente entonces, tiene que sustituirse por ese descendiente.
- Si el elemento a borrar tiene los dos descendientes, entonces tiene que sustituirse por el nodo que se encuentra más a la izquierda en el sub Árbol derecho o por el nodo que se encuentra más a la derecha en el sub Árbol izquierdo.

Para eliminar un nodo del Árbol balanceado, lo primero que se debe hacer es localizar su posición en el Árbol. Se elimina siguiendo

los criterios establecidos anteriormente y se regresa por el camino de búsqueda calculando el FE de los nodos visitados. Si en alguno de los nodos se viola el criterio de equilibrio, entonces debe reestructurarse el Árbol. El proceso termina cuando se llega hasta la raíz del Árbol.

Cabe aclarar que mientras que en el algoritmo de inserción una vez que era efectuada una rotación podía detenerse el proceso, en este algoritmo debe continuarse puesto que se puede producir más de una rotación en el camino hacia atrás.

2.3.2. Análisis de complejidad de un Árbol AVL

En esta parte se analizará cuanto puede crecer como máximo un Árbol AVL, es decir, la altura en el peor de los casos. Este dato nos dará una idea del costo de las operaciones de inserción y eliminación.

Se define el factor de balance como el alto del sub Árbol derecho menos el alto del sub Árbol izquierdo. Entonces en un Árbol AVL, todos los nodos cumplen la propiedad de tener valores del factor de balance iguales a: -1, 0, ó +1.

Sea nh el mínimo número de nodos en un Árbol AVL de altura h dada, que se encuentra en su peor caso de desbalance, si se agrega un nodo, tal que la nueva altura sea (h+1), dejan de ser AVL.

La Figura 20 muestra dichos Árboles, denominados “Fibonacci”, y los factores de equilibrio de sus nodos, para alturas 0, 1, 2, 3 y 4. Se muestran los casos desbalanceados por la derecha, porque los de la izquierda son especulares.

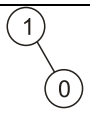
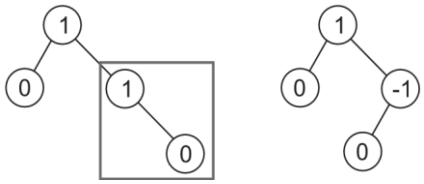
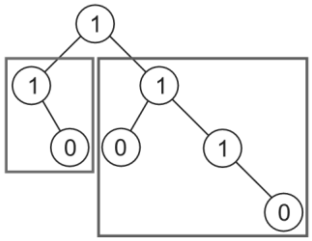
Estado base: n_0= 0	
n_h	*FE de los nodos*
n_1=1	
n_2=2	
n_3= n_2 + n_1+ 1 n_3=4	
n_4= n_3 + n_2+ 1 n_4=7	

Figura 20. Árboles Fibonacci, hasta h=4

Fuente: Elaboración propia

Se cumple que: n_3= n_2 + n_1 + 1, entonces n_3=4

Se observa que se pueden generar 2 Árboles por la derecha y 2 Árboles por la izquierda (especulares) con altura tres (h=3).

Se observa que n_4= 7 porque: n_4= n_3+ n_2+ 1

En la Figura 21, a la raíz se agrega por la derecha un Árbol de Fibonacci de altura 4, y por la izquierda uno de altura 3

$$n_5 = n_4 + n_3 + 1$$

$$n_5 = 12$$

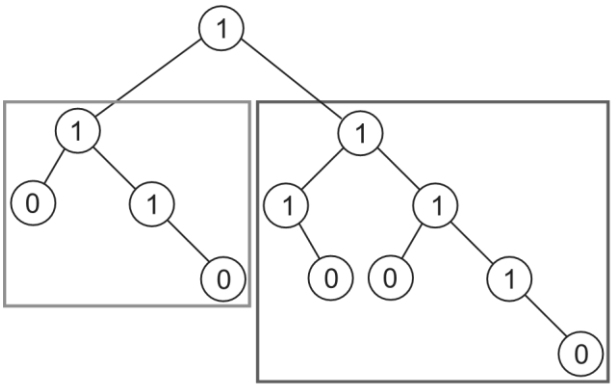


Figura 21. Árbol de Fibonacci para h=5

Fuente: Elaboración propia

Luego podemos decir que n_5 = 12 porque $n_5 = n_4 + n_3 + 1 = 7 + 4 + 1$

Por lo anterior, mediante inducción podemos decir que en general, se tiene la recurrencia:

$$n_h = n_{h-1} + n_{h-2} + 1$$

Donde $n_0 = 0$ y $n_1 = 1$ son las condiciones iniciales.

Si añadimos 1 a ambos lados, obtenemos

$$n_h + 1 = (n_{h-1} + 1) + (n_{h-2} + 1)$$

Así, los números $n_h + 1$ son números de Fibonacci ($F_h = F_{h-1} + F_{h-2}$) (Puntambekar, 2009, págs. 11-4)

Observando las dos secuencias de números que generan n(h) y F(h)

h	0	1	2	3	4	5	6	7
n(h)	0	1	2	4	7	12	20	33
F(h)	0	1	1	2	3	5	8	13
F(h+2)	1	2	3	5	8	13	21	34

Se tiene:

$$n(h) = F(h+2) - 1$$

Usando la fórmula Moivre para los números Fibonacci,

$$F_h = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^h - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^h$$

obtenemos:

$$n_h = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+2} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{h+2} - 1$$

Debido a que $\left|\frac{1}{2}\left(1-\sqrt{5}\right)\right| \approx 0{,}618034$, el segundo término de esta ecuación rápidamente disminuye con el aumento de h y tiene el valor máximo de 0,17082 para h=0, por consiguiente,

$$n_h \geq \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+2} - 0{,}17082 - 1$$

$$n_h \geq \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+2} - 2$$

$$n_h + 2 \geq \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+2}$$

Al tomar log de los dos lados, obtenemos:

$$\lg(n_h + 2) \geq \lg\frac{1}{\sqrt{5}} + 2\lg\left(\frac{1+\sqrt{5}}{2}\right) + h\lg\left(\frac{1+\sqrt{5}}{2}\right)$$

$$\lg(n_h + 2) \approx 0{,}22787 + 0{,}69424h$$

A partir de los cual, obtenemos un límite superior en *h*

$$h \leq 1{,}44042\lg(n_h + 2) - 0{,}32824$$

y por tanto:

$$\lg(n_h + 1) \leq h \leq 1{,}44042\lg(n_h + 2) - 0{,}32824$$

Fórmula para obtener el término n-ésimo de la sucesión de Fibonacci, es también llamada por otros autores como fórmula de Binet.

Luego, la altura del peor caso de un Árbol AVL con n nodos es aproximadamente $1{,}44 \times \log_2(n+2) - 0{,}32$ en el peor caso.

Por poner un ejemplo parecido al que se hizo con el Árbol perfectamente balanceado, si 10 000 elementos se almacenan en un Árbol AVL, entonces el Árbol, en el peor de los casos, tiene una altura de $1{,}44 \times \log_2(10\ 002) - 0{,}32 = 18{,}81 \approx 19$. Esto significa que si 10 000 elementos se almacenan en un Árbol AVL, entonces se revisarán, en el peor de los casos, 19 nodos para encontrar un elemento en particular.

En cuanto al procedimiento de inserción, las pruebas experimentales indican que el rebalanceo es necesario cada dos inserciones, siendo igualmente probables las rotaciones simples y dobles. Por tanto, el Árbol AVL es tan satisfactorio como el Árbol perfectamente balanceado, siendo además mucho más fácil de mantener.

En relación con el procedimiento de eliminación, debe destacarse el hecho de que mientras la inserción de un nodo puede

producir como máximo una rotación, cuando se elimina un nodo, la eliminación puede necesitar una rotación en cada nodo de la trayectoria de búsqueda. En principio este dato resulta alarmante. Sin embargo, los resultados empíricos muestran que sólo es necesario el rebalanceo en una de cada cinco eliminaciones, es decir, el 75% de los casos de eliminación no requieren balanceo en lo absoluto. Por otra parte, sólo el 53% de las inserciones no sacan al Árbol de balance (Karlton, 1976). Por consiguiente, la eliminación que consume más tiempo, ocurre con menos frecuencia que el rebalanceo de los Árboles AVL.

CAPÍTULO III

MARCO METODOLÓGICO

3.1. Caracterización o tipo del diseño de investigación

El diseño de la presente investigación es No experimental – transversal de nivel descriptivo.

3.2. Materiales y/o instrumentos

Equipos:

- ✓ Un ordenador con procesador de alta potencia de cálculo.
- ✓ Una impresora
- ✓ Materiales de escritorio

Software:

- ✓ Lenguaje de programación de alto nivel C++
- ✓ Software estadístico Minitab
- ✓ Software para procesamiento de texto

3.3. Población y muestra

La población de datos contenidos en registros estará conformada por un total de 1583 códigos de postulantes según el formato generado por el sistema de inscripción de los postulantes que se preparan en el Centro Preuniversitario de la Universidad Nacional Jorge Basadre

Grohmann, Tacna. Tomamos esta cantidad de registros basándonos en el número total de postulantes registrados en el proceso CEPU invierno 2015. De esta población tomaremos una muestra que la calculamos de la siguiente manera ya que se trata de una población finita:

$$n = \frac{N \times Z^2 \times p \times q}{e^2 \times (N-1) + Z^2 \times p \times q}$$

Los datos a considerar son los siguientes:

N: Tamaño de la población 1 583

p: 50% Probabilidad a favor.

q: (1- p) = 50% Probabilidad en contra.

Z: Nivel de confianza es 95,0 % el valor correspondiente es 2

e: 10% = 0,1

$$n = \frac{1\,583 \times 2^2 \times 0,5 \times 0,5}{0,1^2 \times (1\,635-1) + 2^2 \times 0,5 \times 0,5} = \frac{1\,583}{17,34} = 91,29$$

Entonces, el tamaño de la muestra que debemos considerar es de 92 registros.

3.4. Tratamiento de datos

Para el tratamiento de los datos se utilizó la estadística descriptiva y las pruebas de contrastación de hipótesis. Se utilizó como herramienta el software Minitab para el procesamiento de los datos.

Se desarrolló una aplicación computacional para la generación automática de registros que contengan un campo de datos denominado código, el cual posee la misma estructura que se le asigna como código de un postulante a la Universidad Nacional Jorge Basadre Grohmann (UNJBG).

Este campo de código está formado por seis caracteres: los dos primeros caracteres están referidos al código de la Escuela Académico profesional a la que postula y los otros cuatro caracteres se forman por el correlativo de las fichas de matrícula del registro de postulantes de la UNJBG.

A continuación mostramos la información que utilizamos para generar la codificación automática del campo código de los registros.

Tabla 4.

Código de las Carreras Profesionales de la UNJBG

codi_carr	desc_carr	codi_cana
01	FARMACIA Y BIOQUIMICA	1
02	MEDICINA HUMANA	1
03	OBSTETRICIA	1
04	ENFERMERIA	1
05	ODONTOLOGIA	1
06	MEDICINA VETERINARIA Y ZOOTECNIA	1
07	BIOLOGIA - MICROBIOLOGIA	1
08	INGENIERIA METALURGICA	2
09	INGENIERIA EN INFORMATICA Y SISTEMAS	2
10	FISICA APLICADA	2
11	ARQUITECTURA, URBANISMO	2
12	INGENIERIA QUIMICA	2
13	INGENIERIA MECANICA	2
14	INGENIERIA EN INDUSTRIAS ALIMENTARIAS	2
15	INGENIERIA CIVIL	2
16	INGENIERIA PESQUERA	2
17	AGRONOMIA	2
18	INGENIERIA GEOLOGICA-GEOTECNIA	2
19	INGENIERIA DE MINAS	2
20	EDUCACION: IDIOMA EXTRANJERO, TRADUCTOR E INTERPRE	3
21	EDUCACION: CIENCIAS SOCIALES Y PROMOCION SOCIAL CU	3
22	EDUCACION: CIENCIAS DE LA NATURALEZ, TECNOLOGIA	3
23	EDUCACION: LENGUAJE, LITERATURA Y GESTION EDUCATIV	3
24	DERECHO Y CIENCIAS POLITICAS	3
25	CIENCIAS DE LA COMUNICACION	3
26	EDUCACION: MATEMATICA, COMPUTACION E INFORMATICA	3
27	ARTES	3
28	CIENCIAS CONTABLES Y FINANCIERAS	4
29	INGENIERIA COMERCIAL	4
30	ECONOMIA AGRARIA	4
31	CIENCIAS ADMINISTRATIVAS	4
32	INGENIERIA AMBIENTAL	2
33	MATEMATICA	2
34	HISTORIA	3

Fuente: Centro Preuniversitario de la UNJBG, 2014

Donde:

codi_carr : Código de la Carrera Profesional

desc_carr : Descripción de la Carrera Profesional

codi_cana : Código de Canal de la Carrera Profesional

Hemos desarrollado una aplicación computacional en el lenguaje de programación C++ versión 5,02 para la generación en forma aleatoria de la población de 1 583 postulantes. Este dato se basa en la información brindada por el Centro Preuniversitario de la UNJBG, ciclo invierno 2 015.

De esta población se toman 10 muestras de 92 postulantes, siendo sólo el campo clave Código de Postulante y el canal al que postula el que se almacena en las estructuras estáticas del tipo Array lineal y la estructura dinámica no lineal del tipo Árbol AVL.

La interfaz de la aplicación implementada para el desarrollo de la tesis es como se muestra en la figura 22.

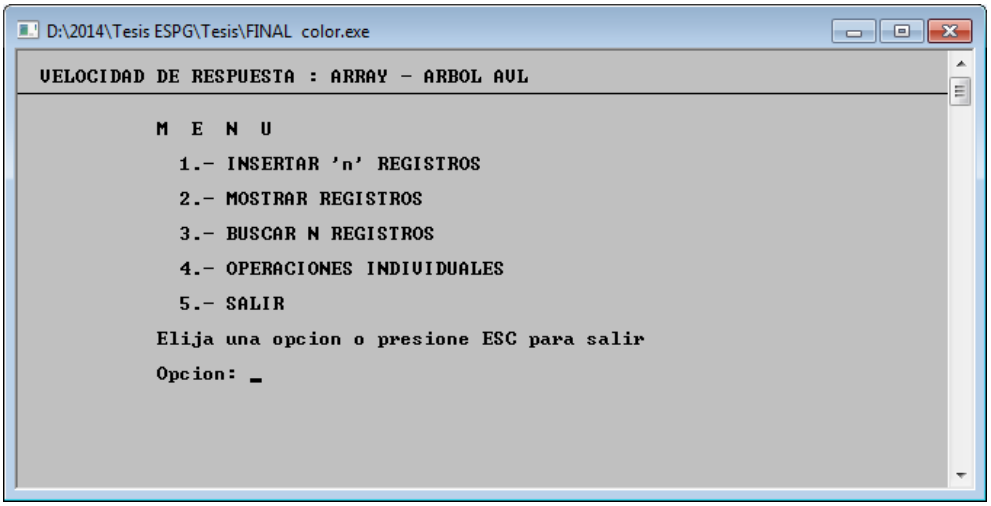


Figura 22. Menú del programa utilizado para generar los registros

Fuente: Elaboración propia

Con la opción 1 generamos los 1 583 registros almacenando en las estructuras de datos los códigos y el canal del postulante.

Para la implementación de la aplicación en C++ se utilizó la librería windows.h. Esta librería contiene dos funciones para cronometrar los tiempos de respuesta de los módulos: QueryPerformanceCounter() que devuelve el tiempo del sistema en unidades del contador de alta resolución y la función QueryPerformanceFrequency() que nos da la frecuencia del contador, y esto permite pasar el tiempo a segundos.

CAPÍTULO IV

RESULTADOS

4.1. Análisis de resultados

Se ha generado 3 muestras, cada una de 92 ítems que corresponde al muestreo que se hace de una población de 1 583 registros de postulantes a la UNJBG. Se genera el CÓDIGO del postulante que consta de 6 caracteres. Estos registros se almacenan en estructuras de datos estáticas tipo Array o Arreglo unidimensional y en estructuras de datos dinámicas del tipo Árbol AVL.

En la siguiente figura se muestra los datos generados y que se almacenan también en un archivo de texto para el posterior procesamiento y análisis estadístico.

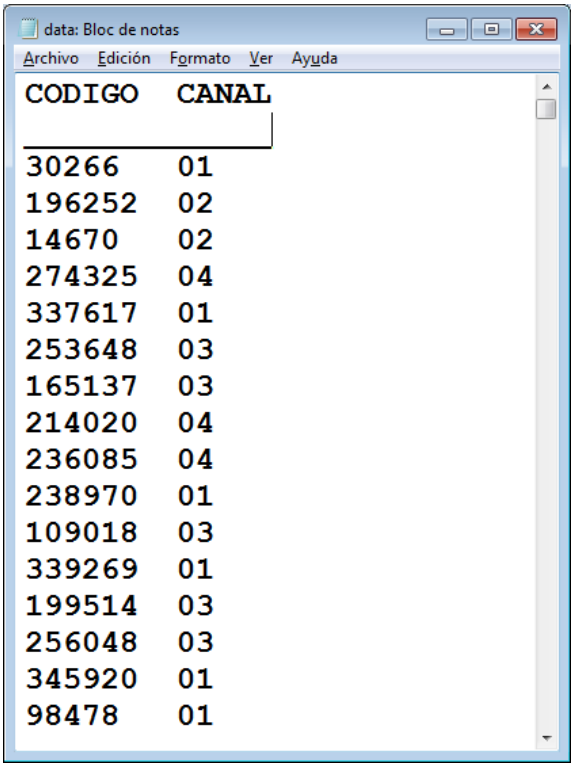


Figura 23. Datos generados y almacenados en un archivo de texto

Fuente: Elaboración propia

Para validar la no redundancia de datos utilizamos la moda como estadístico y obtuvimos los resultados que confirman que no existen códigos repetidos.

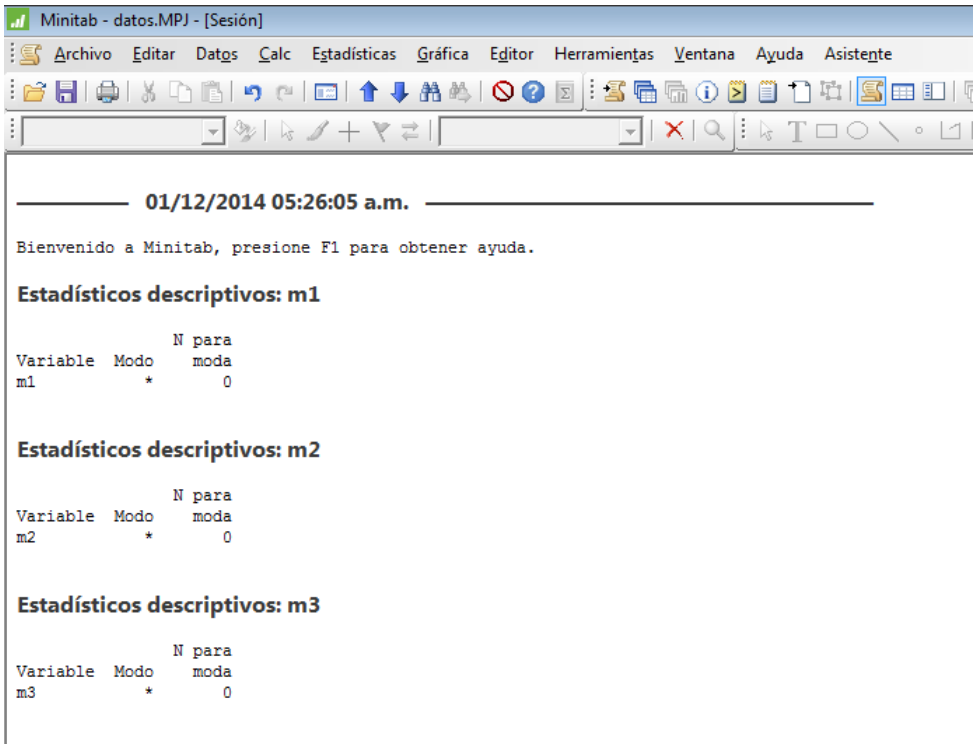


Figura 24. Prueba de consistencia de datos generados

Fuente: Elaboración propia

Resumen de las pruebas de no repetición de códigos utilizando el software Minitab v17.

Estadísticos descriptivos: m1

Variable	Modo	N para moda
m1	*	0

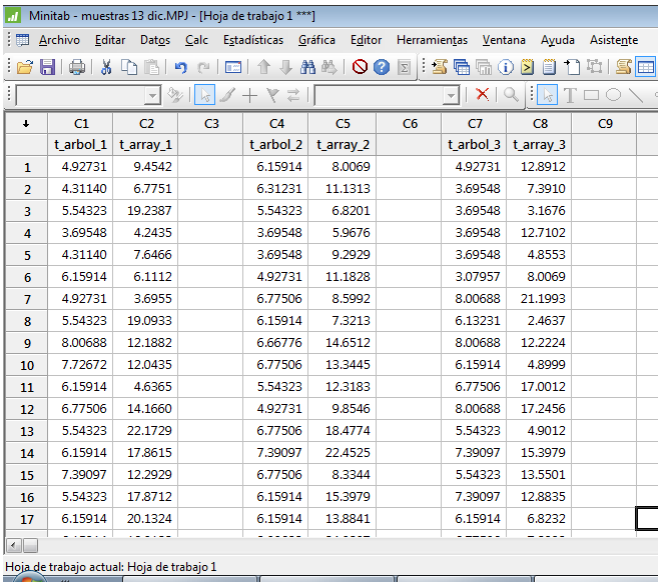
Estadísticos descriptivos: m2

Variable	Modo	N para moda
m2	*	0

Estadísticos descriptivos: m3

Variable	Modo	N para moda
m3	*	0

A partir de estos datos generados que son en total 1 583 registros, se toman 3 muestras de tamaño 92 cada una. Los resultados obtenidos son:



	C1	C2	C3	C4	C5	C6	C7	C8	C9
	t_arbol_1	t_array_1		t_arbol_2	t_array_2		t_arbol_3	t_array_3	
1	4.92731	9.4542		6.15914	8.0069		4.92731	12.8912	
2	4.31140	6.7751		6.31231	11.1313		3.69548	7.3910	
3	5.54323	19.2387		5.54323	6.8201		3.69548	3.1676	
4	3.69548	4.2435		3.69548	5.9676		3.69548	12.7102	
5	4.31140	7.6466		3.69548	9.2929		3.69548	4.8553	
6	6.15914	6.1112		4.92731	11.1828		3.07957	8.0069	
7	4.92731	3.6955		6.77506	8.5992		8.00688	21.1993	
8	5.54323	19.0933		6.15914	7.3213		6.13231	2.4637	
9	8.00688	12.1882		6.66776	14.6512		8.00688	12.2224	
10	7.72672	12.0435		6.77506	13.3445		6.15914	4.8999	
11	6.15914	4.6365		5.54323	12.3183		6.77506	17.0012	
12	6.77506	14.1660		4.92731	9.8546		8.00688	17.2456	
13	5.54323	22.1729		6.77506	18.4774		5.54323	4.9012	
14	6.15914	17.8615		7.39097	22.4525		7.39097	15.3979	
15	7.39097	12.2929		6.77506	8.3344		5.54323	13.5501	
16	5.54323	17.8712		6.15914	15.3979		7.39097	12.8835	
17	6.15914	20.1324		6.15914	13.8841		6.15914	6.8232	

Figura 25. Datos correspondientes a tres muestras de 92 registros Correspondiente al tiempo de respuesta en la búsqueda de datos.

Fuente: Elaboración propia

Aplicando el software Minitab efectuamos los cálculos estadísticos para las 3 muestras, obteniéndose los siguientes resultados:

Estadísticos descriptivos: t_arbol_1, t_array_1, t_arbol_2, t_array_2, t_arbol_3, t_array_3

Variable	Media	Dsv.Est.	Varianza	CoefVar	Mediana	Modo	N para moda
t_arbol_1	7,028	1,028	1,057	14,63	7,391	7,39097	22
t_array_1	12,917	5,064	25,645	39,20	12,768	*	0
t_arbol_2	6,4435	0,9467	0,8962	14,69	6,2717	6,15914, 6,77506	22
t_array_2	11,912	5,055	25,553	42,44	11,045	*	0
t_arbol_3	6,322	1,152	1,327	18,22	6,159	6,15914	17
t_array_3	11,870	4,729	22,366	39,84	11,126	*	0

CAPÍTULO V

DISCUSIÓN DE RESULTADOS

5.1. Interpretación de resultados:

Muestra 1: Para la primera muestra observamos que el tiempo promedio de búsqueda de un registro almacenado en un Árbol AVL (7,028ms) es menor que el tiempo promedio que tarda en localizarlo en un Array lineal (12,917ms).

Muestra 2: Para la primera muestra observamos que el tiempo promedio de búsqueda de un registro almacenado en un Árbol AVL (6,4435ms) es menor que el tiempo promedio que tarda en localizarlo en un Array lineal (11,912ms).

Muestra 3: Para la primera muestra observamos que el tiempo promedio de búsqueda de un registro almacenado en un Árbol AVL (6,322ms) es menor que el tiempo promedio que tarda en localizarlo en un Array lineal (11,870ms).

Respecto a la Desviación Estándar de los datos podemos observar:

Muestra 1: En promedio, los datos se alejan 1,028 unidades respecto a la media del tiempo de búsqueda en Árboles AVL. Así mismo, en promedio, los datos se alejan 5,064 unidades respecto a la media del tiempo de búsqueda en Arrays lineales.

Muestra 2: En promedio, los datos se alejan 0,9467 unidades respecto a la media del tiempo de búsqueda en Árboles AVL. Así mismo, en promedio, los datos se alejan 0,055 unidades respecto a la media del tiempo de búsqueda en Arrays lineales.

Muestra 3: En promedio, los datos se alejan 1,152 unidades respecto a la media del tiempo de búsqueda en Árboles AVL. Así mismo, en promedio, los datos se alejan 4,729 unidades respecto a la media del tiempo de búsqueda en Arrays lineales.

Del estadígrafo Coeficiente de Variación, observamos que los datos se encuentran más dispersos en los Arrays con respecto a los Árboles AVL. Esto se explica por la secuencia lineal en que se encuentran los datos en los Arrays y los datos se almacenan de acuerdo al orden de llegada a la estructura. Sin embargo en los Árboles AVL por no ser lineal su estructura podemos descartar grandes volúmenes de claves en una sola comparación.

Tiempos de demora para insertar las claves en las estructuras:

Respecto al tiempo de demora en insertar las claves en los registros tenemos:

Estadísticos descriptivos: t_ins_AVL, t_ins_array

Variable	Conteo total	Media	Desv.Est.	Varianza	CoefVar	Mediana
t_ins_AVL	15	16 728	1 809	3 272 960	10,82	17 135
t_ins_array	15	4656	799	638 436	17,16	4486

En promedio, el tiempo de inserción de 1 583 claves en Árboles AVL es de 16 728 milisegundos y para insertarlos en el Array es de 4 656 milisegundos. Claramente es más rápido el proceso de inserción de datos en los Arrays aunque esto no implica mayor velocidad de respuesta en los procesos de búsqueda de estos registros tal como se muestra en el análisis anterior.

5.2. Prueba de Hipótesis

Aplicamos la prueba de comparación de medias para los datos de las 03 muestras de los tiempos de respuesta en el proceso de búsqueda de datos contenidos en un Árbol AVL y un Array unidimensional. Utilizamos el software Minitab v17 y obtuvimos los siguientes resultados:

Muestra 1:

Ho: El tiempo promedio de respuesta para la búsqueda de registros localizados en un Árbol AVL es igual al tiempo promedio de respuesta para la búsqueda de registros localizados en un Array lineal.

$$Ho:\ \mu_{t\_AVL} = \mu_{t\_Arr}$$

Ha: El tiempo promedio de respuesta para la búsqueda de registros localizados en un Árbol AVL es menor al tiempo promedio de respuesta para la búsqueda de registros localizados en un Array lineal.

$$Ha:\ \mu_{t\_AVL} < \mu_{t\_Arr}$$

Aplicando la prueba t para 2 muestras, con un nivel de significancia del α = 5% y un nivel de confianza del 95%, se obtiene:

Prueba T e IC de dos muestras: t_arbol_1, t_array_1

```
T de dos muestras para t_arbol_1 vs. t_array_1

                                  Error
                                estándar
                                   de la
            N  Media  Desv.Est.     media
t_arbol_1  92   7,03       1,03      0,11
t_array_1  92  12,92       5,06      0,53

Diferencia = μ (t_arbol_1) - μ (t_array_1)
Estimación de la diferencia:  -5,890
Límite superior 95% de la diferencia:  -4,999
Prueba T de diferencia = 0 (vs. <): Valor T = -10,93
Valor p = 0,000  GL = 182
```

Ambos utilizan Desv.Est. agrupada = 3,6539

Al ser el valor de p = 0,000 menor que nuestro α = 0,050 se rechaza Ho y se acepta Ha. Es decir, el tiempo promedio de respuesta para la búsqueda de registros localizados en un Árbol AVL es menor al tiempo promedio de respuesta para la búsqueda de registros localizados en un Array lineal.

Muestra 2:

Ho: El tiempo promedio de respuesta para la búsqueda de registros localizados en un Árbol AVL es igual al tiempo promedio de respuesta para la búsqueda de registros localizados en un Array lineal.

$$Ho\colon\ \mu_{t\_AVL} = \mu_{t\_Arr}$$

Ha: El tiempo promedio de respuesta para la búsqueda de registros localizados en un Árbol AVL es menor al tiempo promedio de respuesta para la búsqueda de registros localizados en un Array lineal.

$$Ha\colon\ \mu_{t\_AVL} < \mu_{t\_Arr}$$

Aplicando la prueba t para 2 muestras, con un nivel de significancia del α = 5% y un nivel de confianza del 95%, se obtiene:

Prueba T e IC de dos muestras: t_arbol_2, t_array_2

```
T de dos muestras para t_arbol_2 vs. t_array_2

                                    Error
                                  estándar
                                     de la
            N  Media  Desv.Est.     media
t_arbol_2  92  6,444      0,947     0,099
t_array_2  92  11,91       5,05      0,53


Diferencia = μ (t_arbol_2) - μ (t_array_2)
Estimación de la diferencia:  -5,468
Límite superior 95% de la diferencia:  -4,582
Prueba T de diferencia = 0 (vs. <): Valor T = -10,20
Valor p = 0,000  GL = 182
Ambos utilizan Desv.Est. agrupada = 3,6366
```

Al ser el valor de p = 0,000 menor que nuestro α = 0,050 se rechaza Ho y se acepta Ha. Es decir, el tiempo promedio de respuesta para la búsqueda de registros localizados en un Árbol AVL es menor al tiempo promedio de respuesta para la búsqueda de registros localizados en un Array lineal.

Muestra 3:

Ho: El tiempo promedio de respuesta para la búsqueda de registros localizados en un Árbol AVL es igual al tiempo promedio de respuesta para la búsqueda de registros localizados en un Array lineal.

$$Ho:\ \mu_{t\_AVL} = \mu_{t\_Arr}$$

Ha: El tiempo promedio de respuesta para la búsqueda de registros localizados en un Árbol AVL es menor al tiempo promedio de respuesta para la búsqueda de registros localizados en un Array lineal.

$$Ha:\ \mu_{t\_AVL} < \mu_{t\_Arr}$$

Aplicando la prueba t para 2 muestras, con un nivel de significancia del α = 5% y un nivel de confianza del 95%, se obtiene:

Prueba T e IC de dos muestras: t_arbol_3, t_array_3

```
T de dos muestras para t_arbol_3 vs. t_array_3

                                  Error
                                estándar
                                   de la
            N  Media  Desv.Est.    media
t_arbol_3  92   6,32       1,15     0,12
t_array_3  92  11,87       4,73     0,49


Diferencia = μ (t_arbol_3) - μ (t_array_3)
Estimación de la diferencia:  -5,548
Límite superior 95% de la diferencia:  -4,709
Prueba T de diferencia = 0 (vs. <): Valor T = -10,93
Valor p = 0,000  GL = 182
Ambos utilizan Desv.Est. agrupada = 3,4418
```

Al ser el valor de p = 0,000 menor que nuestro α = 0,050 se rechaza Ho y se acepta Ha. Es decir, el tiempo promedio de respuesta para la búsqueda de registros localizados en un Árbol AVL es menor al tiempo promedio de respuesta para la búsqueda de registros localizados en un Array lineal.

Por los resultados observados en las tres muestras anteriores concluimos que el tiempo promedio de respuesta para la búsqueda de registros localizados en un Árbol AVL es menor al tiempo promedio de respuesta para la búsqueda de registros localizados en un Array lineal.

Para el caso de los tiempos de demora en la inserción de registros en las dos estructuras bajo estudio, planteamos las siguientes hipótesis.

Ho: El tiempo promedio de inserción de registros en un Árbol AVL es igual al tiempo promedio de inserción en un Array lineal.

$$Ho:\ \mu_{AVL} = \mu_{Arr}$$

Ha: El tiempo promedio de inserción de registros en un Árbol AVL mayor al tiempo promedio de inserción en un Array lineal.

$$Ha:\ \mu_{AVL} > \mu_{Arr}$$

Aplicando la prueba t para 2 muestras, con un nivel de significancia del α = 5% y un nivel de confianza del 95%, se obtiene:

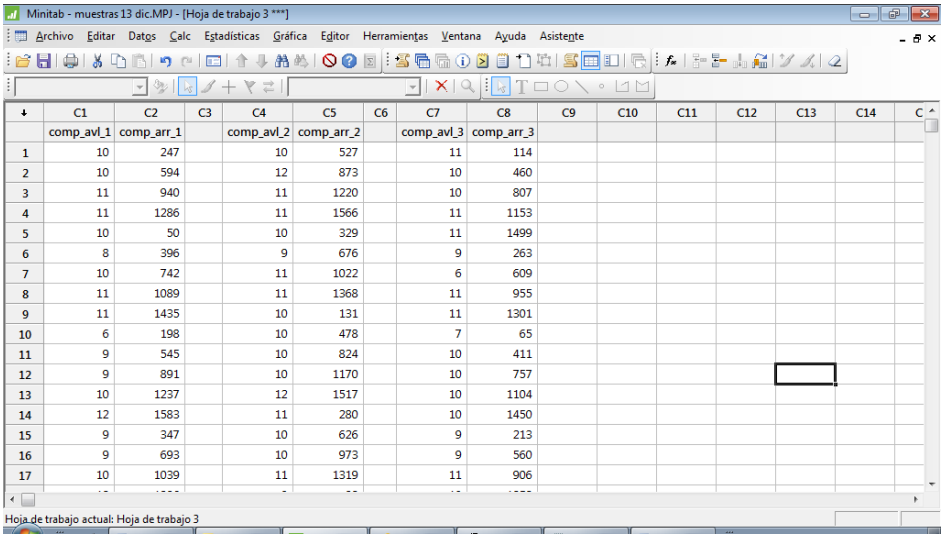
Prueba T e IC de dos muestras: t_ins_AVL, t_ins_array

```
T de dos muestras para t_ins_AVL vs. t_ins_array

              N  Media     Desv.Est.
t_ins_AVL    15  16728       1 809
t_ins_array  15   4656         799

Diferencia = μ (t_ins_AVL) - μ (t_ins_array)
Estimación de la diferencia:  12 072
```

```
Límite inferior 95% de la diferencia:  11 203
Prueba T de diferencia = 0 (vs. >): Valor T = 23,64
Valor p = 0,000  GL = 28
Ambos utilizan Desv.Est. agrupada = 1 398,4627
```

Al ser el valor de p = 0,000 menor que nuestro α = 0,050 se rechaza *Ho* y se acepta *Ha*. Es decir, el tiempo promedio de inserción de registros en un Árbol AVL mayor al tiempo promedio de inserción en un Array lineal.

Respecto al número de comparaciones necesarias para localizar una clave dentro de las dos estructuras de datos tenemos:



	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C
	comp_avl_1	comp_arr_1		comp_avl_2	comp_arr_2		comp_avl_3	comp_arr_3							
1	10	247		10	527		11	114							
2	10	594		12	873		10	460							
3	11	940		11	1220		10	807							
4	11	1286		11	1566		11	1153							
5	10	50		10	329		11	1499							
6	8	396		9	676		9	263							
7	10	742		11	1022		6	609							
8	11	1089		11	1368		11	955							
9	11	1435		10	131		11	1301							
10	6	198		10	478		7	65							
11	9	545		10	824		10	411							
12	9	891		10	1170		10	757							
13	10	1237		12	1517		10	1104							
14	12	1583		11	280		10	1450							
15	9	347		10	626		9	213							
16	9	693		10	973		9	560							
17	10	1039		11	1319		11	906							

Figura 26. Número de comparaciones para localizar un dato almacenado en estructuras de dato tipo Árbol AVL y Array lineal

Fuente: Elaboración propia

Todos los datos obtenidos se muestran en el anexo 1.

Los estadígrafos para estos datos son:

Estadísticos descriptivos: comp_avl_1, comp_arr_1, comp_avl_2, comp_arr_2, comp_avl_3, comp_arr_3

Variable	Conteo total	Media	Desv.Est.	Varianza	CoefVar	Mediana
comp_avl_1	92	9,859	1,280	1,639	12,99	10,000
comp_arr_1	92	783,4	449,9	202403.6	57,43	758,0
comp_avl_2	92	9,870	1,528	2.334	15,48	10,000
comp_arr_2	92	759,9	454,3	206422,4	59,79	756.5
comp_avl_3	92	9,772	1,597	2,552	16,35	10,000
comp_arr_3	92	794,7	459,4	211066,0	57,81	788,5

De estos resultados observamos que en promedio, el número de comparaciones necesarias para localizar un dato contenido en un Array lineal es mucho mayor al número de comparaciones necesarias en Árboles AVL: 9,859 vs 783,4 ; 9,870 vs 759,9 ; 9,772 vs 794,7.

También se observa que los datos referidos al número de comparaciones en arreglos lineales están mucho más dispersos que los referidos al número de comparaciones necesarias en Árboles AVL. Esto se observa en el coeficiente de variación.

Planteamos entonces las siguientes hipótesis estadísticas:

Ho: El número de comparaciones en un Árbol AVL es igual al número de comparaciones en un Array lineal, para localizar un dato.

$$Ho:\ \mu_{c\_AVL} = \mu_{c\_Arr}$$

Ha: El número de comparaciones en un Árbol AVL es menor al número de comparaciones en un Array lineal, para localizar un dato.

$$Ha: \mu_{c\_AVL} < \mu_{c\_Arr}$$

Aplicando la prueba t para 2 muestras, con un nivel de significancia del α = 5% y un nivel de confianza del 95%, se obtiene:

Prueba T e IC de dos muestras: comp_avl_1, comp_arr_1

```
T de dos muestras para comp_avl_1 vs. comp_arr_1

                                    Error
                                estándar
                                   de la
             N  Media  Desv.Est.  media
comp_avl_1  92   9,86       1,28   0,13
comp_arr_1  92    783        450     47

Diferencia = μ (comp_avl_1) - μ (comp_arr_1)
Estimación de la diferencia:  -773,6
Límite superior 95% de la diferencia:  -696,0
Prueba T de diferencia = 0 (vs. <): Valor T = -16,.49
Valor p = 0,000  GL = 182
Ambos utilizan Desv.Est. agrupada = 318,1236
```

Al ser el valor de p = 0,000 menor que nuestro α = 0,050 se rechaza Ho y se acepta Ha. Es decir, el número de comparaciones en un Árbol AVL es menor al número de comparaciones en un Array lineal, para localizar un dato.

CONCLUSIONES

Primera

La velocidad de respuesta en la búsqueda de datos almacenados en memoria está en función del tipo de estructura de datos que utilicemos para contener los datos y al número de comparaciones necesarias para localizar un dato específico

Segunda

El tiempo que se tarda en localizar un dato contenido en una estructura de datos dinámica no lineal del tipo Árbol AVL es menor que el tiempo que se tarda en localizar el mismo dato contenido en una estructura estática del tipo Array lineal: 7,028 milisegundos y 12,917 milisegundos, respectivamente. Pero el tiempo que se requiere para insertar los datos en un Árbol AVL es mayor que el tiempo que se tarda en insertar los mismos datos en un Array lineal: 16 728.00 milisegundos y 4 656.00 milisegundos, respectivamente.

Tercera

El número de comparaciones necesarias para localizar un dato contenido en un Árbol AVL es mucho menor que el número de comparaciones necesarias para localizar el mismo dato en un Arreglo lineal. En promedio 9,859 comparaciones y 783,4 comparaciones respectivamente.

RECOMENDACIONES

Primera

Se recomienda continuar la presente investigación para datos contenidos en otros tipos de Estructuras de Datos dinámicas, como por ejemplo los grafos dirigidos y verificar los tiempos de respuesta de los procesos de búsqueda en esas estructuras.

Segunda

Fomentarlos trabajos de investigación orientados a desarrollar las Ciencias de la Computación para que a partir del entendimiento de las bases teóricas se puedan crear nuevas teorías que la enriquezcan y sean el soporte para el desarrollo de nuevas tecnologías de la información.

REFERENCIAS BIBLIOGRÁFICAS

AHO, A., HOPCROFT, J. & ULLMAN, J. (1998). Estructura de Datos y Algoritmos. New York: Addison-Wesley Iberoamericana.

ALONSO, P. & GARCÍA, F. (1998). Diseño e Implementación de Programas en Lenguaje C. Valencia: Libro Docente.

BISBAL, J. (2009). Anual de Algorítmica. Recursividad, complejidad y diseño de algoritmos. España: Editorial UOC.

BRASSARD, G. & BRATLEY, P. (1997). Fundamentos de algoritmia. México: Pearson Educación.

CAIRÓ, O. & GUARDATI, S. (2007). Estructura de Datos (3ª ed.). México: McGraw-Hill.

CALCEDO, A., WAGNER, G. & MÉNDEZ, R.M. (2010). Introducción a la Teoría de Grafos. Colombia: Ediciones Elizcom.

CERVIGON, C. (2009). Algoritmos evolutivos: un enfoque práctico. España: RA-MA.

DROZDEK, A. (2005). Data Structures and Algorithms in C++ (2ª ed). Massachusetts: Grace Fujimoto.

GARRIDO, A. & FERNÁNDEZ, J. (2006). Estructuras de Datos en C++. Madrid: Delta Publicaciones.

GÓMEZ DE SILVA, G., & ANIA, I. (2008). Introducción a la Computación. Mexico DF: Cengage Learning.

HERNANDEZ, R., LAZARO, J. C., DORMIDO, R., & ROS, S. (2000). Estructura de datos y algoritmos. Madrid: Prentice Hall.

HEILEMAN G. (1999). Estructura de Datos, Algoritmos y Programación Orientada a Objetos. México: Mc Graw Hill.

HERNANDEZ, R., FERNANDEZ, C. & BAPTISTA, P. (2009). Metodología de la Investigación. México: Mc Graw Hill.

JOYANES, L. & ZAHONERO, I. (2003). Programación en C: Metodología, algoritmos y estructura de datos. España: Mc Graw Hill.

JOYANES, L. (2006). Fundamentos de Programación: Algoritmos y Estructura de Datos. México: McGraw-Hill.

JOYANES, L., SANCHEZ, L. & ZAHONERO, I. (2007). Estructura de Datos en C++. España: McGraw-Hill.

KNUTH, D. (2010). The Art of Computer Programming. Fundamental Algorithms. Massachusetts: Addison-Wesley.

KOLMAN, B., BUSBY, R. & ROSS, S. (1997). Estructuras de Matemáticas Discretas para la Computación. México: Prentice Hall Hispanoamericana S.A.

KOWALSKI, S. & MONTGOMERY, D. (2011). Design and Analysis of Experiments. United States of America: Minitab Companion

KRUSE, R. (1996). Estructura de Datos y Diseño de Programas. México: Prentice-Hall Hispanoamericana.

LOOMIS, M. (1999). Estructura de Datos y Organización de Archivos. México: Prentice Hall Hispanoamericana.

MANBER, U. (2001). Introduction to Algorithms: A Creative Approach. New York: Addison-Wesley Iberoamericana.

MATHEWS, P. (2005). Design of experiments with MATLAB. United States of America: Quality press.

PELÁEZ, C. & VISO, E. (2008). Introducción a las Ciencias de la Computación. México: Las prensas de Ciencias - UNAM

ROBERT H. (2004). Doing Data Analysis with Minitab 14. United States of America: Thomson/Brooks.

RODRIGUEZ, M., GONZÁLEZ, P. & GOMEZ, M. (2011). Estructuras de Datos. Un enfoque moderno. Madrid: Editorial Complutense.

SEDGEWICK, R. (1999). Algoritmos en C++. United States of America: Addison Wesley.

TENENBAUM A. & AUGENSTEIN, M. (2004). Estructura de Datos en C. México: Prentice Hall Hispanoamericana.

WINDER, R. (1995). Desarrollo de Software con C++. Madrid: Ediciones Díaz de Santos.

WIRTH, W. (1994). Algoritmos y Estructura de Datos. México: Prentice Hall Hispanoamericana

Información Web:

CAPELLO, D. (2012). Cómo Medir el tiempo de una rutina computacional. Recuperado el 06 de julio de 2014 de:

http://dacap.com.ar/blog/cpp/medir-el-tiempo-de-una-rutina/

DUCH, A. (2007). Algoritmos.

Recuperado el 01 de abril del 2014 de:

http://www.lsi.upc.edu/~duch/home/duch/analisis.pdf

GURIN, S. (2004). Árboles AVL.

Recuperado el 25 setiembre del 2014 de:

http://es.tldp.org/Tutoriales/doc-programacion-arboles-avl/avl-trees.pdf

ANEXOS

Anexo 1: Lista registros generados

Codificación de postulantes en el Centro Preuniversitario de la UNJBG

CODIGO	CANAL										
274552	02	216028	03	165900	01	180005	04	120783	03	096768	04
321873	02	017878	03	102504	02	324170	04	129528	03	164156	02
166547	02	335552	03	198912	04	091630	01	348040	04	238952	01
196264	01	230612	01	074030	02	109522	01	225352	03	096570	01
013450	03	104865	04	032102	03	168674	01	340858	02	096136	03
323820	03	117260	01	117440	04	149316	04	315169	02	153448	01
161784	03	156560	04	124044	03	026110	04	164710	04	268412	02
028750	01	315789	01	282150	02	126000	04	048280	01	047093	03
285824	03	151011	01	025663	03	312381	02	232784	02	170800	03
267686	01	342424	03	243587	02	035175	01	111384	01	092228	04
135400	02	251538	03	025440	02	070144	04	064708	03	075740	02
077514	03	206678	03	318752	03	129381	04	061858	04	293468	02
029529	04	224420	02	066752	04	241128	04	024098	03	215690	03
274064	04	133260	02	136559	01	127311	01	017825	02	241764	02
292528	03	090440	02	273875	01	158093	02	198291	03	252000	04
159395	01	219406	04	040910	03	294800	04	023536	01	246730	02
056854	04	078582	03	051316	01	249924	01	161056	01	022960	01
344540	02	029760	02	087750	04	117365	02	052496	03	179700	01
011040	01	255640	01	197915	04	346454	02	029120	02	251383	04
168172	03	272155	03	246000	03	178640	01	275146	02	257581	03
031827	04	265958	01	287840	03	320226	04	027298	02	273740	03
037505	01	192925	02	348810	04	156060	01	131515	02	261755	02
334960	02	313402	02	111078	04	341012	02	159760	02	184000	04
140663	04	178897	04	136980	03	151840	01	261244	02	286797	03
320448	02	177296	02	068437	01	112592	01	337144	04	090258	03
011501	04	259595	02	138322	02	095040	02	138162	03	104031	01
068194	02	081966	03	169826	02	085792	04	106568	04	296260	01
283800	03	170240	01	310317	02	132065	04	016034	02	047248	02
284368	03	274232	03	298565	03	060529	04	295726	03	167233	04
235065	02	137942	02	054478	04	175050	01	241212	02	193188	03
132792	02	203910	03	326495	02	234542	04	057428	01	221410	03
111328	01	131339	02	084520	01	326765	01	011218	01	245718	03
328400	03	152823	02	282240	03	346804	02	196614	01	068741	02
170524	02	129930	03	187200	02	197014	02	193577	01	325571	01
322520	03	082616	02	075155	02	290640	02	033125	02	090960	02
270275	03	149408	02	059256	04	263688	01	054826	03	063070	02
176792	02	261878	03	318269	03	337519	02	228706	04	321807	01
212884	03	126837	01	267700	02	068872	02	231406	03	068649	04
016549	01	261496	01	117768	04	034944	03	347214	01	335644	04
139875	04	026320	04	138010	02	334740	01	313394	04	252820	01
120826	02	265266	03	231996	02	105600	01	276360	04	197647	03
065375	01	281930	04	151300	03	329280	01	224800	01	195291	03
196540	01	133895	04	076256	02	075416	04	153752	04	285500	01
011558	03	313200	03	321745	01	319556	04	076981	02	170117	01
291909	01	027198	02	209620	02	223958	04	333216	04	338625	04
307066	03	073612	01	083542	03	306460	02	333403	01	054340	04
123456	02	322286	04	313520	01	202582	01	021984	03	135875	01
297472	04	100527	03	194944	04	253923	02	081680	04	316379	02
196344	02	296695	04	172568	01	156585	01	185462	02	207976	04
144600	02	036322	03	147256	01	033680	02	161445	02	108648	03
112955	01	065397	04	244011	01	170373	04	334856	02	252448	01
233295	01	172210	04	260445	03	043543	02	015724	04	239194	04
154280	01	064848	03	213586	03	308442	04	082814	02	091696	04
261140	01	023075	02	315771	01	061036	01	331325	01	326704	04
274100	02	023616	03	092253	03	154143	03	143926	04	150714	04
040408	04	015120	03	226160	04	107793	01	243920	04	245018	03
103972	02	207168	03	177383	03	291704	04	265896	01	320824	01
		222558	01	219840	03	216341	03	222288	01	108360	03

235783 03 213942 02 152420 02 081095 03 086628 01 071590 02
164512 04 171008 02 335002 03 326923 03 157290 02 059498 04
083520 02 192844 02 110536 01 042480 04 091380 03 262612 04
260816 02 043960 02 024000 04 181675 04 095263 03 035416 02
222836 01 318704 02 222258 04 060928 04 063140 01 068005 04
057484 02 048062 04 035163 03 134115 02 213031 04 274336 01
341924 03 038780 04 208451 02 166224 02 267412 04 045156 04
174384 03 149716 03 348400 01 218635 01 093904 01 184088 03
128426 01 342930 03 067217 01 191420 03 288022 01 133945 01
015545 03 215476 01 295268 04 132566 03 091749 04 280138 04
347663 03 299782 04 145710 04 322945 04 290145 01 279315 04
020482 02 339380 03 054279 01 050800 02 124012 02 191603 02
056918 02 246876 04 119546 02 084780 02 295572 03 013580 03
017907 02 183112 01 339685 03 057136 03 216290 04 238435 02
189212 02 129346 01 110105 04 179808 01 238190 03 155184 01
316860 03 191086 01 311000 04 308928 01 136010 02 053072 02
023593 03 280340 01 100690 02 245290 04 166341 01 295760 02
315882 04 280700 04 125448 02 080152 01 215915 01 304000 02
163566 03 286810 03 162240 03 109658 02 016240 03 148781 01
170880 02 186153 04 024384 03 135330 02 177526 02 176806 03
057110 04 323823 04 079268 04 070490 04 278486 04 114324 01
060794 03 283275 04 254940 03 100330 04 216588 04 195722 04
153072 01 288073 04 104298 04 192957 01 290927 01 085888 03
248395 03 156927 02 153179 04 094856 03 090816 01 124220 01
083475 02 013885 02 238394 02 128433 03 126766 01 108974 01
329408 02 221392 04 098560 01 283592 02 282688 01 146712 02
217867 03 248240 02 147200 04 295636 03 215824 01 116532 04
201628 03 034130 02 218853 03 191036 03 260728 04 283540 03
218736 01 077104 01 257420 03 286286 01 339544 03 017919 03
084032 02 179815 03 093500 03 251115 04 290439 04 155899 01
298860 04 111048 02 223824 02 052995 03 265590 03 059965 03
090500 02 162875 04 345872 04 016180 04 325745 04 122319 01
084406 03 097292 02 074577 01 096656 03 348300 02 314164 01
346250 01 162101 03 326428 04 290120 03 059568 01 234888 04
262976 04 304494 03 097890 04 215510 01 037932 04 324099 03
202538 02 303037 02 173628 02 195081 02 062864 03 074655 01
013592 02 110851 01 146976 03 030520 02 038080 02 265704 01
203570 02 212744 04 213540 01 133551 03 030801 02 309626 01
174659 03 276280 03 278638 03 035371 04 289254 04 094667 03
117532 01 033096 03 260339 04 328788 04 055335 03 211062 02
156292 03 011532 04 116513 02 347480 02 151312 04 094072 02
021338 01 028870 01 195500 04 320485 03 286498 02 270378 01
186876 03 209118 02 192299 04 238805 01 192528 02 042294 02
203786 01 121526 03 310668 01 082565 02 079160 01 069680 03
241623 03 058810 02 128990 04 196750 01 338240 02 170320 03
064524 03 278475 01 097608 03 266154 02 035412 04 289472 02
168440 03 317790 01 184485 04 154704 01 282400 01 231375 01
347152 03 142624 02 119973 02 245520 04 197690 04 130490 01
204410 04 236858 04 311678 02 114383 03 225955 03 320850 03
216710 01 071656 01 272543 04 330485 02 303734 02 233590 03
048728 02 035408 03 226335 03 271811 03 330225 04 073771 02
300000 03 125696 02 152422 04 107682 03 196847 03 090445 01
083436 03 327500 03 130081 02 066318 04 058240 03 335888 01
171530 04 295765 01 122956 04 020056 03 258052 01 295235 03
317988 02 291633 01 077960 01 229807 02 327739 03 269568 02
298504 02 124336 01 168450 02 038518 01 270160 04 245346 02
084794 04 204765 01 017144 01 153416 04 034839 02 305606 02
269177 02 256025 01 256256 01 068048 04 314828 02 224850 02
193270 04 090560 02 076700 04 070760 03 135616 01 048878 02
265120 01 115500 03 145486 03 259403 01 084000 03 251216 01
253232 02 133324 01 211088 04 210900 04 241384 03 065708 02
062768 01 047152 01 040416 01 158176 02 325827 01 072241 04
327212 03 258628 04 301111 02 047520 01 185200 04 115800 02
092794 02 054776 03 284378 01 060650 04 239642 02 252592 03

248686	04	247120	01	253186	01	348412	03	255496	04	240408	04
142179	04	270852	01	051288	04	252332	03	112110	02	250782	02
027088	02	149250	01	170104	02	101855	02	321320	02	216598	04
142963	01	248592	01	325992	02	053100	04	270367	04	228480	03
078400	01	162981	02	060655	03	038900	04	145040	03	190050	03
091276	02	236675	01	334830	04	226220	03	106875	02	289875	01
032264	02	250273	04	069874	02	095261	03	145764	04	219057	01
226245	01	187092	04	319108	03	119308	04	140768	03	143224	01
112277	03	083004	01	112722	03	101980	04	020412	01	317679	04
286848	01	119536	03	341788	02	034559	04	212313	01	135366	03
284229	04	211396	02	240125	03	260385	02	105708	02	047850	02
225400	03	289348	03	259461	01	342086	02	346644	03	123576	01
039858	02	241560	03	092540	03	079736	04	127863	03	176400	01
171640	03	236885	04	326400	02	133000	02	213200	02	339920	02
058600	04	192864	01	081450	03	047762	02	166820	03	287216	01
089745	01	250440	03	093100	01	332040	01	193326	02	208390	04
248848	03	165844	01	320784	04	195686	04	111320	04	135240	02
190185	02	312819	04	165432	02	244866	02	271583	04	308651	04
221918	02	236748	04	077772	02	268250	01	340161	01	332731	04
122720	03	210736	01	122390	04	111609	02	052584	02	310208	01
331896	03	015210	01	294704	03	319274	02	025998	01	087075	04
280870	02	010956	04	167904	01	119389	03	024810	02	292177	03
062090	02	322818	02	137649	02	291235	03	056450	04	147854	01
115440	02	144025	03	279349	01	237000	02	223234	01	181274	04
230972	02	266504	02	132502	02	113912	01	278887	01	186956	03
310515	02	213370	04	150914	04	276755	01	018933	03	085092	01
334600	01	089648	01	017071	01	094876	03	081926	01	105118	02
183267	04	339528	01	238420	02	209815	01	197012	03	311640	04
014480	01	261004	02	120446	03	090293	02	054312	03	309642	03
227776	04	302716	01	217822	02	299750	01	215838	01	114752	04
330384	03	298368	04	208880	01	349781	01	183930	02	171172	01
024290	03	165060	02	256050	02	023453	02	342022	03	244689	01
150972	02	332058	03	253632	01	059468	03	207214	03	071959	03
265528	04	205180	02	265620	04	042472	02	189239	03	152204	01
190095	01	267454	02	150630	03	166229	04	299485	01	172952	03
266580	01	289230	02	058705	02	070462	01	120675	04	260949	02
120184	02	293320	02	349756	02	084180	02	274939	03	267343	03
099480	02	104678	02	173882	02	315420	03	276257	03	010512	02
152778	04	259672	02	158382	01	229397	01	178276	04	026572	04
339630	02	130752	03	296011	02	115617	02	223770	02	060930	02
174022	03	323078	01	161463	04	299400	04	128072	04	224628	03
115512	03	325440	04	211475	04	170632	01	137212	03	036842	04
064320	02	110698	02	250159	02	174992	04	167532	04	031435	01
062600	02	247586	03	021336	02	210768	03	074695	04	346036	01
280878	03	292274	02	237517	02	066657	02	011378	04	276675	01
295847	03	126544	02	276640	01	178613	02	023440	03	148806	03
218248	02	233389	02	038742	02	202525	01	241008	03	246560	01
169580	04	240450	03	218775	01	142420	04	349804	04	045360	04
211360	01	292616	01	308896	02	109361	04	236220	03	095095	02
079297	01	311444	01	302216	02	306670	03	330232	02	228060	01
314276	04	112350	02	090517	04	281585	02	038200	03	152368	02
251355	04	161011	01	106375	03	215140	02	279730	02	146608	04
149405	01	076745	04	056588	02	206899	01	049000	01	063593	03
112756	04	086788	04	131875	01	017612	02	096413	01	084860	03
312524	04	309060	02	231632	03	247739	04	166815	04	156305	03
274260	03	034888	04	347812	04	091192	01	079676	04	191000	03
338532	04	064750	03	182084	04	223790	02	208508	02	324960	04
038004	01	226096	03	130400	01	292120	04	343072	02	317347	04
320368	04	300640	04	158416	04	292540	04	164704	01	134700	04
261215	03	325833	02	208425	03	327728	02	019268	03	236336	02
055590	04	243776	02	340155	03	306750	01	082426	03	256297	01
203856	04	104985	01	297057	03	118014	02	158788	03	287954	02
318626	02	159154	02	275477	04	335020	04	121436	02	329568	04
126736	04	173475	02	078292	03	342078	04	277886	03	34060	03

204569	03	194726	02	222300	04	074003	04	264388	03	345984	03
318384	04	250950	02	333592	04	095640	04	087132	01	051157	02
323940	01	136456	02	104825	01	171854	03	227122	01	331140	04
239892	01	049800	01	286878	01	154276	03	329840	01	314670	02
342241	02	248829	04	092478	01	030156	01	176188	02	349587	02
39865	04	281890	03	329660	02	058357	03	187608	03	040433	03
74865	04	336600	01	279867	02	025296	01	244577	04	077016	03
186587	02	116480	04	233469	03	135260	03	084348	03	161548	02
189574	01	281597	02	152356	04	270634	02	308875	02	210200	01
079286	02	334040	01	149876	04	298592	03	316040	03	013008	03
095866	03	166108	04	050602	04	075820	04	106632	03	330190	04
230160	03	115612	02	331972	03	325994	01	029452	01	304305	04
162428	02	272104	04	053300	04	303272	03	233722	03	137420	01
315986	03	110788	04	318490	01	180616	04	075200	03	275398	04
036632	01	097953	04	159467	03	175574	03	168644	02	346320	01
275080	02	208439	02	059481	01	076496	04	261456	01	097340	03
312997	04	215108	04	089512	03	136920	03	331895	04	215500	02
157950	04	088620	03	047051	01	273595	03	016164	03	217997	02
146840	02	067916	03	157816	02	260446	01	145318	04	013504	03
132600	03	176660	04	210370	03	035781	01	252420	02	140690	04
249060	01	063320	02	223250	02	235364	02	228375	04	218614	04
307684	02	244388	04	183049	01	090839	03	286834	02	076972	02
046410	02	057820	02	240128	01	264670	01	268100	03	147674	01
131418	03	164825	01	151920	03	157752	01	150579	01	211182	02
104706	03	305608	02	212430	01	191040	02	221400	02	270928	03
251862	01	043671	04	063056	04	025305	01	096712	02	126378	02
058534	01	172464	01	123264	01	295836	03	036116	03	283255	02
073032	02	049567	03	154000	01	154048	03	263420	02	325780	01
098468	04	205400	01	312232	03	189102	01	287025	02	267224	02
099743	02	032950	03	192968	02	014612	03	274392	01	278928	03
094071	02	197806	02	063175	02	111800	03	179004	04	090390	02
296016	01	107736	01	088864	01	099215	04	059700	02	256864	03
290474	02	102018	01	332200	01	098608	01	223742	03	328504	02
185213	01	267568	04	260988	03	287440	02	142210	04	011858	03
256004	01	049840	02	212350	02	335440	03	026231	03	063872	04
296749	03	288502	01	080000	03	072128	02	299760	04	036113	04
319088	01	209990	03	276720	03	303926	02	190428	02	260488	04
208633	03	185708	01	090138	04	301175	01	289328	02	119175	03
162056	03	287946	04	203215	02	148384	03	216520	02	127960	03
326060	04	347984	03	041888	04	320812	03	157488	04	091825	03
025745	02	018610	01	269003	02	296320	01	120365	01	321257	04
146602	02	040660	01	104111	03	183985	04	125300	01	339560	03
219016	01	266612	02	185855	04	071400	03	160224	01	027639	03
221856	02	332815	02	083050	04	294764	03	147120	04	153762	01
314376	03	203608	03	311504	02	192500	04	322721	03	237360	01
113975	02	224300	02	307662	03	137980	01	314784	03	338298	03
099918	03	335068	02	320065	03	295548	01	140060	04	209818	03
307769	02	314400	01	188618	04	333305	02	106967	02	190036	01
309746	02	267452	02	270956	01	059387	02	054675	01	085053	01
028449	03	284690	04	066824	01	121394	02	305176	03	319300	01
257250	02	293748	01	119124	01	149916	01	322525	01	132992	04
073678	02	314576	01	241153	04	262169	04	251335	02	212460	04
110256	03	107440	04	335659	02	248106	04	141609	03	336062	03
049548	03	231700	02	040874	01	014536	01	232554	03	107588	02
280826	01	274576	02	095798	02	206240	04	047782	02	072356	03
026430	01	273331	01	069245	03	299746	01	167280	02	097468	02
098082	02	288176	01	105338	01	192360	01	314577	02	034624	03
209740	03	278992	02	296350	03	111688	01	116820	01	348032	01
250752	03	249190	01	136960	01	282764	04	024512	01	297360	02
163074	03	139920	04	304812	04	240780	03	344304	02	173058	04
051234	03	087104	02	069803	03	216002	03	124643	01	349790	01
225165	04	144480	02	061976	01	167788	03	339244	01	180960	03
215264	03	236002	02	121182	03	038968	02	335481	02	199132	03
085674	02	285137	04	231628	03	348287	01	312666	03	038892	01

```
105688  03    175754  02    286544  02    167745  03    213314  01    176709  03
102423  02    046130  04    265716  01    190000  03    335199  01    116904  01
013865  03    218040  03    157920  03    235786  01    321440  03    085770  02
158901  04    333340  04    065187  02    043562  03    222185  02    146122  04
297066  04    343070  02    026647  03    073462  03    196756  03    335515  03
240791  04    204352  02    081284  02    327327  03    336980  01    162050  02
162054  02    263690  03    202292  02    103502  03    072728  01    230360  03
259673  03    170209  01    064696  01    041120  01    277585  04    293424  04
066215  04    151368  04    119467  01    058478  01    240512  03    037205  01
015756  04    176390  01    251750  04    277835  01    189360  02    171040  04
203945  03    109395  03    160720  02    268275  02    281028  03    125050  02
130644  01    123432  03    073597  04    073275  03    011583  01    236166  04
122365  01    153952  01    172758  03    257186  01    085380  04    154615  01
212000  02    326080  04    186590  01    159616  04    158375  04    180146  03
```

```
Total : 1583 registros
Tiempo insercion arreglo: 4355.85 miliseg
Tiempo insercion arbol: 14457.7 miliseg
```

Anexo 2: Búsquedas y tiempo de respuesta de 92 registros (tamaño de muestra).

MUESTRA 1

N°	cod_busc	t_arbol	t_array
1	201288	4.92731	9.45421
2	182469	4.3114	6.77506
3	28352	5.54323	9.23871
4	175680	3.69548	4.24352
5	75880	4.3114	7.64663
6	120587	6.15914	6.11121
7	103320	4.92731	3.69548
8	156464	5.54323	19.0933
9	145291	8.00688	12.1882
10	312424	7.72672	12.0435
11	20340	6.15914	4.63653
12	225336	6.77506	14.166
13	158538	5.54323	22.1729
14	70526	6.15914	17.8615
15	166214	7.39097	12.2929
16	31552	5.54323	17.8712
17	275268	6.15914	20.1324
18	162421	6.15914	16.0122
19	258114	6.77506	9.85463
20	300492	8.00688	15.3979
21	338000	8.00688	16.5636
22	37253	8.6228	14.6277
23	34776	7.55343	27.5132
24	96400	6.15914	14.6363
25	88584	7.39097	22.7888
26	148008	7.38882	14.1121
27	87696	8.6228	12.8991
28	11092	8.00688	3.59912
29	145071	7.39097	10.6712
30	219604	6.77506	9.79921
31	179032	8.00688	11.7024
32	154500	6.77506	7.11933
33	309920	8.6228	15.2872
34	222070	6.15914	6.10091
35	287228	7.39097	8.6228
36	256928	6.15914	24.2817
37	330533	6.77506	27.1002
38	341804	6.77506	14.1435
39	36944	8.6228	14.7132
40	106102	7.39097	11.0865
41	341444	8.00688	19.7093
42	235596	7.39097	7.54663
43	308325	8.00688	16.5342
44	185256	7.39097	19.1829
45	39025	7.39097	27.3721
46	233037	6.77506	6.21291
47	254777	6.77506	15.2099
48	288275	6.77506	16.1321
49	330684	7.39097	16.8473
50	74270	8.00688	13.4241
51	145248	7.39097	24.6366
52	224440	6.15914	4.88473
53	42330	8.00688	27.7161
54	166576	6.77506	8.16118
55	306376	7.39097	16.6252
56	50104	8.00688	26.0391
57	213227	6.77506	12.2994
58	187168	7.39097	26.4843
59	139202	6.13543	9.55321
60	28215	5.54323	4.88463
61	246100	7.39097	12.0023
62	37956	8.00688	16.2551
63	68357	8.6228	23.4047
64	179625	7.39097	12.5023
65	306592	7.39097	10.4121
66	111923	8.00688	9.92437
67	38838	7.39097	36.9548
68	275830	6.77506	8.12001
69	280490	8.00688	13.6277

70	288576	6.77506	8.94112
71	78710	7.39097	16.5536
72	303970	8.6228	17.2456
73	78850	8.00688	16.0138
74	148610	6.77506	14.6465
75	147245	7.39097	9.88211
76	237886	8.00688	16.5452
77	278676	8.00688	16.1882
78	275271	6.77506	16.2441
79	304305	8.00688	20.3252
80	123868	5.54323	14.0912
81	21820	7.39097	5.54323
82	65497	7.39097	8.63745
83	108368	7.39097	10.3991
84	243888	6.15914	8.00688
85	238679	6.15914	7.39097
86	323750	6.77506	4.81212
87	112175	7.39097	13.5501
88	225960	6.77506	10.4705
89	159066	8.6228	6.00991
90	288272	6.77506	9.34232
91	228360	6.77506	12.9342
92	206210	6.77506	11.7127

Busquedas : 92 de 1583

M U E S T R A 2

N°	cod_busc	t_arbol	t_array
1	266081	6.15914	8.00688
2	326376	6.31231	11.13133
3	127912	5.54323	6.82011
4	334955	3.69548	5.967611
5	291539	3.69548	9.29288
6	198026	4.92731	6.18281
7	237280	6.77506	8.59923
8	70980	6.15914	7.3213
9	90750	6.66776	14.65121
10	229236	6.77506	13.34452
11	342885	5.54323	12.3183
12	177534	4.92731	9.85463
13	256136	6.77506	18.4774
14	186651	7.39097	6.45251
15	154031	6.77506	8.3344
16	196559	6.15914	15.3979
17	177949	6.15914	13.88412
18	53214	8.00688	24.0207
19	57672	6.15914	11.00387
20	334110	6.15914	7.8734
21	70720	8.00688	9.88711
22	118403	7.39097	5.54323
23	104594	7.39097	10.07878
24	199616	6.77506	6.15914
25	68736	6.15914	4.29911
26	20572	6.15914	6.69992
27	244552	4.3114	8.6228
28	331660	6.77506	10.12425
29	165430	5.54323	12.65614
30	127867	6.77506	6.59121
31	124794	6.15914	19.10021
32	136150	6.77506	12.29975
33	97063	6.15914	19.0933
34	150100	5.54323	10.12444
35	111020	5.54323	7.32312
36	201637	6.77506	4.64323
37	267872	5.26526	3.612
38	257780	6.77506	11.48711
39	286468	8.00688	11.12239
40	75180	6.15914	14.61342
41	162421	8.00688	19.80067
42	52634	6.77506	19.69988
43	145576	5.54323	13.5501
44	187168	8.6228	19.71209
45	81588	7.39097	11.0865
46	56574	8.00688	20.9411
47	129325	7.39097	11.32452
48	16873	8.00688	16.0138
49	68357	7.39097	14.42442
50	269883	8.6228	10.39996

51	165536	6.77506	5.9934
52	12456	5.54323	15.66721
53	346910	6.15914	17.2456
54	269080	4.92731	9.99726
55	228360	4.92731	17.8615
56	136314	5.54323	6.6625
57	204445	6.15914	6.69912
58	135993	7.39097	9.66552
59	261912	4.92731	7.2993
60	34682	6.77506	7.4002
61	151952	6.77506	16.76129
62	123120	7.39097	7.39097
63	293523	6.56999	6.8177
64	121713	6.77506	14.23235
65	80281	7.39097	15.29991
66	341444	6.77506	14.7819
67	26308	7.39097	9.23772
68	61798	5.99923	3.69548
69	211654	6.77506	8.87322
70	260714	6.15914	16.6297
71	262644	6.15914	6.9992
72	278676	7.39097	8.24892
73	178430	6.15914	8.01341
74	241817	6.23112	6.12123
75	236362	5.91121	6.56627
76	120516	5.54323	4.9992
77	155873	6.15914	9.38788
78	195840	6.13424	3.65121
79	48422	6.15914	9.23199
80	118863	6.77506	12.76901
81	215792	6.01987	4.3114
82	229892	5.54323	5.98817
83	15164	6.77506	10.54002
84	258300	7.39097	9.98923
85	50091	6.15914	15.65266
86	136355	6.77506	6.67882
87	90970	6.77506	8.12789
88	181027	6.15914	7.42552
89	66053	7.39097	8.4032
90	133061	6.77506	9.03211
91	102716	6.15914	7.9267
92	11593	6.15914	3.63652

Busquedas : 92 de 1583

M U E S T R A 3

N°	cod_buscad	t_arbol	t_array
1	271240	4.92731	12.89121
2	304146	3.69548	7.39097
3	312432	3.69548	3.16757
4	255440	3.69548	8.71022
5	133690	3.69548	4.85529
6	294728	3.07957	8.00688
7	205144	8.00688	9.19932
8	245550	6.13231	2.46366
9	184169	8.00688	12.22236
10	107410	6.15914	4.89991
11	149086	6.77506	17.00121
12	303970	8.00688	17.2456
13	238970	5.54323	4.90121
14	258633	7.39097	15.3979
15	295796	5.54323	13.5501
16	179625	7.39097	12.88354
17	332180	6.15914	6.82321
18	310296	6.77506	7.29981
19	104620	6.77506	13.22328
20	185521	4.92731	9.81187
21	190434	5.54323	11.12981
22	236556	7.39097	16.0138
23	91728	7.39097	9.20013
24	333752	6.77506	17.8615
25	25380	7.39097	6.09972
26	208940	8.00688	10.31291
27	262768	6.15914	8.53493
28	41176	6.77506	8.59945
29	86594	4.92731	9.31208
30	260371	4.3114	4.3114
31	306502	5.54323	6.77506

32	331695	6.15914	3.07957
33	343450	5.009921	8.59932
34	267036	6.77506	14.166
35	121690	7.39097	12.18232
36	249148	4.92731	14.11725
37	239325	6.77506	7.32716
38	72024	7.39097	16.55931
39	26308	8.00688	12.9342
40	64932	6.099891	5.49981
41	27226	7.39097	11.01665
42	77170	6.77506	9.25543
43	319158	4.92731	7.29996
44	205512	6.77506	8.01566
45	78850	4.92731	10.74865
46	153763	4.92731	11.10085
47	144025	6.77506	8.6228
48	289695	4.92731	11.99238
49	336192	5.54323	11.7024
50	231980	6.15914	16.6297
51	241710	4.92731	6.70231
52	148610	6.15914	15.29919
53	124310	5.54323	12.29901
54	83650	6.72663	7.32887
55	63261	7.39097	11.00921
56	283136	5.54323	13.11565
57	111020	5.54323	3.69548
58	113475	6.15914	7.89321
59	39138	6.77506	11.10021
60	208432	7.39097	9.85223
61	251362	4.92731	12.3183
62	109760	7.39097	9.17232
63	292140	6.15914	9.21332
64	16748	7.39097	15.35246
65	100920	7.288912	14.10915
66	300608	8.00688	11.88782
67	221256	6.69945	10.39923
68	38670	8.00688	8.59645
69	121760	5.54323	6.52325
70	102849	8.00688	6.13432
71	207180	6.15914	8.70021
72	120516	6.15914	4.88932
73	341304	6.15914	9.65212
74	131120	8.00688	4.89223
75	95208	7.39097	6.15914
76	263370	6.15914	15.29912
77	100296	6.77506	14.7819
78	274954	6.15914	6.67375
79	215971	6.15914	8.71095
80	337881	6.34534	11.0865
81	343754	5.54323	9.44911
82	35050	6.77506	8.14933
83	316384	6.15914	7.28979
84	13292	6.15914	5.65191
85	123868	6.15914	7.29912
86	77048	6.77506	8.61211
87	183946	7.39097	15.31021
88	45087	5.99121	5.54323
89	51548	7.39097	8.53645
90	141790	6.77506	10.4705
91	56605	8.00688	11.12199
92	269017	7.23217	9.85463

Busquedas : 92 de 1583

Anexo 3: Reporte de número de comparaciones

para localizar datos contenidos en Árbol AVL y en Array lineal

DATOS 1

No.	cod_busc	comp_avl	comp_arr
0	324240	10	247
1	148924	10	594
2	36340	11	940
3	165750	11	1286
4	120608	10	50
5	61912	8	396
6	305598	10	742
7	246056	11	1089
8	113588	11	1435
9	33771	6	198
10	272650	9	545
11	105610	9	891
12	102564	10	1237
13	347683	12	1583
14	218900	9	347
15	311454	9	693
16	41878	10	1039
17	113100	12	1386
18	116848	10	149
19	205604	10	495
20	71719	10	842
21	234815	9	1188
22	212840	12	1534
23	22450	8	298
24	156640	9	644
25	115708	11	990
26	55040	10	1336
27	91253	10	100
28	252688	10	446
29	218869	10	792
30	198848	11	1139
31	246331	10	1485
32	12470	9	248
33	76264	9	595
34	159520	10	941
35	44212	10	1287
36	242845	9	397
37	121072	9	743
38	98720	10	328
39	162160	11	674
40	190544	9	1020
41	262943	12	1367
42	48014	11	130
43	248265	10	476
44	182950	9	823
45	237720	10	1169
46	250186	11	1515
47	284050	8	278
48	50622	10	625
49	314460	11	971
50	145662	12	1317
51	53614	11	81
52	222600	10	427
53	305402	10	773
54	215159	8	1120
55	243698	10	1466
56	154973	10	229
57	264885	9	576
58	161232	8	922
59	265898	12	1268
60	38584	10	31
61	145523	10	378
62	338385	11	724
63	161220	9	1070
64	96789	11	1417
65	201611	11	180
66	142500	9	526
67	262300	12	873
68	268732	11	1219
69	310366	11	1565
70	242500	10	675
71	182710	10	1021
72	267482	10	131
73	176515	10	477
74	286608	10	1170
75	69732	12	1516
76	34760	8	279
77	290800	10	626
78	217445	10	972
79	260957	11	1318
80	282060	8	428
81	143978	9	774
82	43738	11	1467
83	165279	8	230
84	229180	7	923
85	281159	11	1269
86	75808	7	32
87	214242	6	379
88	175504	9	725
89	333076	10	1071
90	187408	9	181
91	144092	10	527

DATOS 2

No.	cod_busc	comp_avl	comp_arr
0	144092	10	527
1	262300	12	873
2	188496	11	1220
3	71520	11	1566
4	183192	10	329
5	333906	9	676
6	37800	11	1022
7	129570	11	1368
8	267482	10	131
9	313616	10	478
10	198792	10	824
11	286608	10	1170
12	245514	12	1517
13	144070	11	280
14	290800	10	626
15	311088	10	973
16	110977	11	1319
17	111926	9	82
18	274941	9	429
19	324156	9	775
20	52736	12	1121
21	43738	11	1467
22	281678	9	231
23	238902	10	577
24	229180	7	923
25	134600	11	1270
26	107696	8	33
27	214242	6	379
28	201008	9	726
29	54688	11	1072
30	48132	12	1418
31	224706	8	182
32	256935	10	528
33	84284	10	874
34	185236	5	112
35	300020	7	459
36	188778	8	805
37	123688	11	1151
38	286150	11	1498
39	121702	8	261
40	239274	8	607
41	138622	11	954
42	310140	11	1300
43	134490	5	63

44	260580	10	409
45	237130	10	756
46	164600	10	1102
47	239544	10	1448
48	346072	10	212
49	50800	11	558
50	318880	11	904
51	235530	10	1251
52	111596	11	14
53	150311	9	360
54	212304	11	707
55	179444	10	1053
56	236215	11	1399
57	96820	11	162
58	175632	8	509
59	38630	11	855
60	58989	11	1201
61	222460	11	1548
62	155072	8	311
63	135253	11	657
64	340160	11	1004
65	283794	11	1350
66	97262	8	113
67	306978	10	460
68	345276	10	806
69	241800	10	1152
70	228163	9	262
71	298915	10	608
72	97881	11	1301
73	60620	6	64
74	138151	10	410
75	219844	10	757
76	198590	11	1103
77	102997	11	1449
78	305520	8	559
79	82591	11	905
80	174430	8	15
81	154520	11	361
82	238295	9	1054
83	288288	11	1400
84	87112	9	163
85	65887	13	510
86	347935	11	856
87	320780	11	1202
88	17588	8	312
89	257700	10	658
90	242310	9	1351
91	320545	11	114

DATOS 3

No.	cod_busc	comp_avl	comp_arr
0	320545	11	114
1	306978	10	460
2	111020	10	807
3	254391	11	1153
4	302600	11	1499
5	261193	9	263
6	126501	6	609
7	46560	11	955
8	97881	11	1301
9	237268	7	65
10	16135	10	411
11	219844	10	757
12	76756	10	1104
13	39828	10	1450
14	184224	9	213
15	315694	9	560
16	105680	11	906
17	10658	10	1252
18	174430	8	15
19	27914	11	362
20	33264	11	708
21	238295	9	1054
22	237336	11	1401
23	30108	7	164
24	65887	13	510
25	102048	7	857
26	227150	9	1203
27	107884	11	1549
28	110230	9	313
29	164212	9	659
30	166504	10	1005
31	213694	11	243
32	162356	10	590
33	286875	10	936
34	170015	10	1282
35	236075	8	46
36	75624	10	392
37	345024	8	738
38	158148	9	1085
39	169474	11	1431
40	197200	10	194
41	53404	12	541
42	101332	8	887
43	251356	12	1233
44	229490	10	1579
45	346608	12	343
46	49448	12	689
47	232081	11	1035
48	179544	11	1382
49	173120	9	145
50	61464	9	491
51	65450	11	838
52	342615	12	1184
53	281418	11	1530
54	271680	7	293
55	251672	9	640
56	300862	11	986
57	294262	10	1332
58	125680	8	96
59	179600	8	442
60	315245	10	788
61	52680	11	1135
62	47096	12	1481
63	135674	9	244
64	300360	10	591
65	246120	9	937
66	142400	11	1283
67	306240	6	393
68	110838	9	739
69	299076	11	1432
70	13050	6	195
71	211027	10	888
72	202696	11	1234
73	101812	11	1580
74	118903	11	344
75	283878	10	690
76	221240	11	1036
77	274309	5	146
78	233810	10	492
79	125260	9	1185
80	215621	9	1531
81	78984	10	294
82	38635	10	641
83	297588	9	987
84	179200	10	1333
85	301351	9	443
86	110397	10	789
87	130960	11	1482
88	245842	11	245
89	128115	11	938
90	141950	11	1284
91	333210	5	47

Anexo 3: Código fuente en C++

```
// Velocidad de respuesta de un array y un Árbol AVL
// Descripcion: Muestra el numero de comparaciones para
// una búsqueda en un Array y en un Árbol AVL
#include<conio.h>
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#include<stdio.h>
#include<iomanip.h> //Para manipuladores de datos, autocompletar ceros
#include<windows.h> //Para contar el tiempo

double performancecounter_diff(LARGE_INTEGER *a, LARGE_INTEGER *b)
{
  LARGE_INTEGER freq;
  QueryPerformanceFrequency(&freq);
  return (double)(a->QuadPart - b->QuadPart) / (double)freq.QuadPart;
}

struct alumno //estructura para cada elemento del array
{
    long codi;
   int canal;
};

class nodo // estructura para cada nodo del arbol
{
    public:
        int fe; // factor de equilibrio
        long codi;
       int canal;
        nodo *izq,*der;
    public:
        void carga(nodo *raiz);
        void inserta_avl(nodo *&raiz,struct alumno,int &cen);
        void elimina_avl(nodo *&,int , int &);
        void nodo::reestructura1(nodo *&raiz,int&cen);
        void nodo::reestructura2(nodo *&raiz,int&cen);
        void nodo::borrar(nodo *&aux,nodo *&q,int&cen);
        void preorden(nodo *raiz);
        void inorden(nodo *raiz);
        void postorden(nodo *raiz);
        int nodo::busca_avl(nodo *raiz, int clave);
        nodo * nuevo_nodo(struct alumno);
        int altura(nodo *raiz);
        int nodo::nodos(nodo *raiz); //devuelve la cantidad de nodos del arbol
        void nodo::muestra_nivel(nodo *raiz,int n);
        int nodo::nivel(nodo *raiz,int clave);
        void nodo::peso(nodo *raiz,int &k); //numero de nodos terminales u hojas
};

char si_o_no();
int opcion(int max_op);
// Diseño del Menu Principal
void menu();
// Diseño de rótulos de las tablas
void cabecera();
int busca_array(int clave);
void mygx(int posf);
char msje[30];
```

```
alumno dato,array[2000]; // se instancia el arreglo de registros
int clav; //variable temporal que almacena el valor de cod en la busqueda
int rep; //Flag que indica si el nodo a insertar esta repetido
int tope=-1; //Tope maximo del array
int comp,comp2; //Numero de comparaciones
LARGE_INTEGER t_ini, t_fin; //Variables para medir tiempo
double secs, cont1, cont2; //Variables que almacenan las unidades de tiempo


void main(void)
{
   system("color 70");
   textbackground(7);
   clrscr();
    nodo arbol,*raiz=NULL;
    int op,cen,i,k,cent,band,s,control[2000];
    //float prom;
    char c;
    alumno clave;
   ofstream archivo("c:\\espg1\\codigos2.txt");  // se instancia el fichero
   ofstream reporte("c:\\espg1\\comp2.txt");  // se instancia el fichero
   clrscr();
    do
    {
      clrscr();
       strcpy(msje,"VELOCIDAD DE RESPUESTA : ARRAY - ARBOL AVL");
       menu();
       op=opcion(5);
      clrscr();
       switch(op)
       {
          case 1:
          system("cls");
          strcpy(msje,"INSERCION DE REGISTROS EN LAS ESTRUCTURAS DE DATOS");
           cabecera();
           //char str[5];
             long num; //Para almacenar el numero aleatorio
             int tot;  //Total de datos aleatorios
           //float ad;
           int can;
       int a; //variable temporal del tamaño del array
       cout<<" NUMERO TOTAL DE REGISTROS : "; cin>>tot;
       srand(time(NULL));
       a=0;
       cout<<endl;
       cout<<"\tÚÄÄÄÄÄÄÄÄÄÄÄÂÄÄÄÄÄÄÄÄÄ¿"<<endl;
       cout<<"\t³ CODIGO    ³ CANAL  ³"<<endl;
       cout<<"\tÃÄÄÄÄÄÄÄÄÄÄÄÅÄÄÄÄÄÄÄÄÄ´"<<endl;
       archivo<<"CODIGO        CANAL  "<<endl;
       archivo<<"_____________________"<<endl;
              cont1=0; cont2=0;
       while(a<tot)
       {
       rep=0;
               do
               {
                  num=rand()*rand() % 350000;
                  if(a>0)
                     for(k=0;k<a;k++)
                         if(array[a].codi==num)
                         cent=1;
               }
               while(num<10000||num>=350000||cent==1);
```

```
    can= 1+rand() % 4;
    dato.codi=num;
    dato.canal=can;
    cen=0; clave=dato;
            QueryPerformanceCounter(&t_ini);
    arbol.inserta_avl(raiz,clave,cen);
            QueryPerformanceCounter(&t_fin);
            secs = performancecounter_diff(&t_fin, &t_ini);
            cont1= cont1+secs;
    if(rep==0)
    {
            QueryPerformanceCounter(&t_ini);
    ope=tope+1;
    array[tope]=dato;
            QueryPerformanceCounter(&t_fin);
            cs = performancecounter_diff(&t_fin, &t_ini);
            cont2=cont2+secs;
           mygx(11);
    cout<<setfill('0')<<setw(5)<<array[tope].codi;
            archivo<<setfill('0')<<setw(5)<<array[tope].codi;
            mygx(23);
    if(int(array[tope].canal/10)==0)
            {
           cout<<"  0"<<array[tope].canal;
            archivo<<"  0"<<array[tope].canal;
            }
    else
             {
                cout<<"  "<<array[tope].canal;
                archivo<<"  "<<array[tope].canal;
              }
    cout<<endl;
            archivo<<endl;
    a=a+1;
    }
}
cout<<"\tÃÄÄÄÄÄÄÄÄÄÄÄÄÄÅÄÄÄÄÄÄÄÄ´"<<endl;
cout<<"\tÀÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÙ"<<endl;
    cout<<endl;
    cout<<"\nTiempo insercion arreglo: "<<cont2*1000000<<" æs";
    cout<<"\n\nTiempo insercion arbol: "<<cont1*1000000<<" æs";
    archivo<<endl;
    archivo<<"\nTotal : "<<tot<<" registros";
    archivo<<"\nTiempo insercion arreglo: "<<cont2*1000000<<" microseg";
    archivo<<"\nTiempo insercion arbol: "<<cont1*1000000<<" microseg";
getch();
    system("cls");
break;
case 2:
    system("cls");
    strcpy(msje,"LISTADO DE REGISTROS");
     cabecera();
cout<<"\t³ CODIGO    ³ CANAL ³"<<endl;
cout<<"\tÃÄÄÄÄÄÄÄÄÄÄÄÄÄÅÄÄÄÄÄÄÄÄ´"<<endl;
arbol.inorden(raiz);
    cout<<"\tÃÄÄÄÄÄÄÄÄÄÄÄÄÄÅÄÄÄÄÄÄÄÄ´"<<endl;
    cout<<"\t³ CODIGO    ³ CANAL ³"<<endl;
    cout<<"\tÀÄÄÄÄÄÄÄÄÄÄÄÄÄÁÄÄÄÄÄÄÄÄÙ"<<endl;
cout<<"\nELEMENTOS EN EL ARRAY: "<<(tope+1)<<endl;
    cout<<"NUMERO TOTAL DE NODOS: "<<arbol.nodos(raiz)<<endl;
    cout<<"ALTURA DEL ARBOL (h): "<<arbol.altura(raiz)<<endl;
    getch();

    system("cls");
```

```
break;
case 3:
strcpy(msje,"BUSCAR N REGISTROS");
cabecera();
    int w,ind, tot1, resp, resp2;
    for(w=1;w<=3;w++)
    {
         if(tope!=-1)
    {
    tot1=92;
    reporte<<"No.    CODIGO     ARBOL      ARRAY"<<endl;
            reporte<<"       BUSCADO      c          c"<<endl;
    for (i=0;i<tot1;i++)
    {
                do
                {
                   band=0;
                   srand(time(NULL));
                   ind = 1 + rand() % (tope+1);
                   if(i>0)
                   for(s=0;s<i;s++)
                       if(ind==control[s])
                         {
                          band=1;
                            s=i;
                        }
    }
           while(band==1);
           control[i]=ind;
    clav=array[ind].codi;
    cout<<" ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
    gotoxy(3,wherey());
            cout<<"CODIGO BUSCADO: "<<array[ind].codi<<"  --> "<<i<<endl;
            reporte<<i<<"\t"<<array[ind].codi;
    comp=0;
    QueryPerformanceCounter(&t_ini);
            resp=arbol.busca_avl(raiz,clav);
    QueryPerformanceCounter(&t_fin);
    secs = performancecounter_diff(&t_fin, &t_ini);
           gotoxy(3,wherey());
           cout<<"ARBOL  : ";
        if (resp==1)
    {
    cout<<comp<<" COMPARACIONES"<<endl;
                   reporte<<"\t"<<comp;
        gotoxy(3,wherey()); cout<<"TIEMPO : "<<secs*1000000<<" æs";
                   }
        else cout<<"****NSE"<<endl;
        comp2=0;
        QueryPerformanceCounter(&t_ini);
                   resp2=busca_array(clav);
        QueryPerformanceCounter(&t_fin);
        secs = performancecounter_diff(&t_fin, &t_ini);
                gotoxy(39,wherey()-1);
                cout<<"ARRAY  : ";
            if (resp2==1)
    {
    cout<<comp2<<" COMPARACIONES"<<endl;
    gotoxy(39,wherey()); cout<<"TIEMPO : "<<secs*1000000<<" æs";
            reporte<<"\t"<<comp2;
    else
                cout<<"NSE";
    cout<<endl;
                reporte<<endl;
```

```
        }
                reporte<<"\nBusquedas : "<<tot1<<" de "<<tot<<"\n\n";;
        }
        else cout<<endl<<" NO HAY ELEMENTOS EN LAS ESTRUCTURAS";
        system("cls");
             }
        break;
        case 4:
             int opc;
    do
    {
    strcpy(msje,"OPERACIONES ESPECIFICAS");
    cabecera();
    gotoxy(13,5);printf("M E N U :");
        gotoxy(15,7);printf("1.- Insertar un registro");
    gotoxy(15,9);printf("2.- Buscar un gistro");
    gotoxy(15,11);printf("3.- Eliminar un registro");
    gotoxy(15,13);printf("4.- Volver al menu");
    gotoxy(13,15);printf("\n\nElija una opcion o presione ESC para salir");
    gotoxy(13,17);printf("Opcion: ");
    opc=opcion(4);
    clrscr();
        switch(opc)
    {
    case 1:
    do
    {
         system("cls");
    strcpy(msje,"INSERTAR 1 REGISTRO");
    cabecera();
    cout<<endl<<"CODIGO: ";
    cin>>dato.codi;
    cout<<"CANAL : "; cin>>dato.canal;
    rep=0; cen=0; clave=dato;
    arbol.inserta_avl(raiz,clave,cen);
    if(rep==0)
    {
        tope=tope+1;
        array[tope]=dato;
    }
    else cout<<"LA CLAVE YA EXISTE ... "<<endl;
    cout<<endl<<"INSERTAR MAS CLAVES?(S/N): ";
         c=si_o_no();
    cout<<endl;
    while(tolower(c)=='s');
    break;
    case 2:
    strcpy(msje,"BUSCAR 1 REGISTRO");
    cabecera();
    int resp, resp2;
         if(tope!=-1)
    {
    cout<<" CLAVE DEL REGISTRO A LOCALIZAR: "; cin>>clav;
    cout<<" ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
    gotoxy(3,wherey());cout<<"EL CODIGO DEL REGISTRO BUSCADO ES: "<<clav<<endl;
    comp=0;
    QueryPerformanceCounter(&t_ini);
        resp=arbol.busca_avl(raiz,clav);
        QueryPerformanceCounter(&t_fin);                        secs =
performancecounter_diff(&t_fin, &t_ini);
    gotoxy(3,wherey()); cout<<"ARBOL: ";
    if (resp==1)
{
cout<<comp<<"\tCOMPARACIONES: "<<endl;
```

```
gotoxy(3,wherey()); cout<<"NIVEL: "<<arbol.nivel(raiz,clav)<<" | TIEMPO:
"<<secs*1000000<<" æs";
}
else cout<<"NSE"<<endl;
comp2=0;
QueryPerformanceCounter(&t_ini);
resp2=busca_array(clav);
QueryPerformanceCounter(&t_fin);
secs = performancecounter_diff(&t_fin, &t_ini);
gotoxy(39,wherey()-1); cout<<"ARRAY: ";
if (resp2==1)
{
cout<<comp2<<" COMPARACIONES"<<endl;
gotoxy(39,wherey()); cout<<"POSICION: "<<(comp2/2)<<" | TIEMPO: "<<secs*1000000<<"
æs";
}
else cout<<"NSE";
cout<<endl;
cout<<" ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
}
else cout<<endl<<" NO HAY ELEMENTOS EN LAS ESTRUCTURAS";
getch();
system("cls");
break;
sase 3:
do
{
    system("cls");
    int clave;
    strcpy(msje,"ELIMINAR 1 REGISTRO");
    cabecera();
    if(tope!=-1)
    {
    cout<<endl<<"CODIGO: "; cin>>clave;
    cen=0;
    QueryPerformanceCounter(&t_ini);
    arbol.elimina_avl(raiz,clave,cen);
    QueryPerformanceCounter(&t_fin);
                            secs = performancecounter_diff(&t_fin, &t_ini);
cout<<endl<<"TIEMPO PARA ARBOL AVL: "<<secs*1000<<" ms";
//Eliminar elemento del arreglo
QueryPerformanceCounter(&t_ini);
if(tope+1>=1)
{
int i=0;
int cen=0;
while(i<=tope+1 && cen==0)
{
if(array[i].codi==clave)
{
        cen=1;
    tope=tope-1;
    for(int j=i;j<tope+1;j++)
    array[j]=array[j+1];
}
}
else
    i++;
}
if(cen==0)
    cout<<"\nARRAY: No se encontro el registro... ";
}
else
cout<<endl<<"ARRAY: El array está vacío.";
```

```
QueryPerformanceCounter(&t_fin);
secs = performancecounter_diff(&t_fin, &t_ini);
cout<<endl<<"TIEMPO PARA EL ARRAY: "<<secs*1000<<" ms"<<endl;
cout<<endl<<"MAS CLAVES?(S/N): ";c=si_o_no();
}
else
{
    cout<<endl<<" NO HAY ELEMENTOS EN LAS ESTRUCTURAS";
    getch(); c='n';
}
} while(tolower(c)=='s');
break;
case 4:
opc=20;
 break;
}
clrscr();
} while(opc!=20);
break;
case 5:
    op=27;
         break;
    }
   }while(op!=27);
   archivo.close();
   reporte.close();
}

int opcion(int max_op)
{
    int c;
   int x=wherex(),y=wherey(); max_op=max_op+48;
    do
    {
        gotoxy(x,y);c=getche();
   }while((c<49||c>max_op)&&c!=27);
    if(c!=27) c=c-48;
    return c;
}

char si_o_no()
{
    char c;int x=wherex(),y=wherey();
    do
    {
        gotoxy(x,y);c=getche();
    }while(tolower(c)!='s'&&tolower(c)!='n');
return c;
}

void cabecera()
{
    gotoxy(3,2);cout<<msje;
    gotoxy(1,3);cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ";
}
void menu()
{
    cabecera();
    gotoxy(13,5);printf("M  E  N  U ");
    gotoxy(15,7);printf("1.- INSERTAR 'n' REGISTROS");
    gotoxy(15,9);printf("2.- MOSTRAR REGISTROS");
    gotoxy(15,11);printf("3.- BUSCAR N REGISTROS");
    gotoxy(15,13);printf("4.- OPERACIONES INDIVIDUALES");
    gotoxy(15,15);printf("5.- SALIR");
```

```
    gotoxy(13,17);printf("Elija una opcion o presione ESC para salir");
    gotoxy(13,19);printf("Opcion: ");
}

//necesita tener un nodo creado.
void nodo::carga(nodo *raiz)
{
    nodo *q;char op;
    cout<<"raiz->codo= ";cin>>raiz->codi;
    cout<<"raiz->fe= ";cin>>raiz->fe;
    cout<<"HAY NODO A LA IZQUIERDA DE "<<raiz->codi<<"?:(S/N) ";op=si_o_no();
    cout<<endl;
    if(tolower(op)=='s')
    {
        q=new(nodo);
        raiz->izq=q;
        carga(raiz->izq);
    }
    else   raiz->izq=NULL;
    cout<<"HAY NODO A LA DERECHA DE "<<raiz->codi<<"?:(S/N) ";op=si_o_no();
    cout<<endl;
    if(tolower(op)=='s')
    {
        q=new(nodo);
        raiz->der=q;
        carga(raiz->der);
    }
    else   raiz->der=NULL;
}

void nodo::inserta_avl(nodo *&raiz,struct alumno clave,int &cen)
{
    nodo *raiz1,*raiz2;
    if(raiz!=NULL)
    {
        if(clave.codi<raiz->codi)
        {
            inserta_avl(raiz->izq,clave,cen);
            if(cen==1)
            {
                switch(raiz->fe)
                {
                case 1:raiz->fe=0;cen=0;break;
                case 0:raiz->fe=-1;break;
                case -1:
                    raiz1=raiz->izq;
                    if(raiz1->fe<=0)
                    {
                        raiz->izq=raiz1->der;
                        raiz1->der=raiz;
                        raiz->fe=0;
                        raiz=raiz1;
                    }
                    else
                    {
                        raiz2=raiz1->der;
                        raiz->izq=raiz2->der;
                        raiz2->der=raiz;
                        raiz1->der=raiz2->izq;
                        raiz2->izq=raiz1;
                        if(raiz2->fe==-1) raiz->fe=1;
                        else   raiz->fe=0;
                        if(raiz2->fe==1)  raiz1->fe=-1;
                        else   raiz1->fe=0;
```

```
                    raiz=raiz2;
                }
                raiz->fe=0;
                cen=0;
                break;
            }
        }
    }
    else if(clave.codi>raiz->codi)
    {
        inserta_avl(raiz->der,clave,cen);
        if(cen==1)
        {
            switch(raiz->fe)
            {
            case -1:raiz->fe=0;cen=0;break;
            case 0:raiz->fe=1;break;
            case 1:
                raiz1=raiz->der;
                if(raiz1->fe>=0)
                {
                    raiz->der=raiz1->izq;
                    raiz1->izq=raiz;
                    raiz->fe=0;
                    raiz=raiz1;
                }
                else
                {
                    raiz2=raiz1->izq;
                    raiz->der=raiz2->izq;
                    raiz2->izq=raiz;
                    raiz1->izq=raiz2->der;
                    raiz2->der=raiz1;
                    if(raiz2->fe==1)  raiz->fe=-1;
                    else   raiz->fe=0;
                    if(raiz2->fe==-1) raiz1->fe=1;
                    else   raiz1->fe=0;
                    raiz=raiz2;
                }
                raiz->fe=0;
                cen=0;
                break;
            }
        }
    }
    else   {rep=1;}
  }
  else
  {
      raiz=nuevo_nodo(clave);
      cen=1;
  }
}

void nodo::elimina_avl(nodo *&raiz, int clave,int & cen)
{
  if(raiz!=NULL)
  {
      if(clave<raiz->codi)
      {
          elimina_avl(raiz->izq,clave,cen);
          reestructura1(raiz,cen);
      }
      else if(clave>raiz->codi)
```

```
        {
            elimina_avl(raiz->der,clave,cen);
            reestructura2(raiz,cen);
        }
        else
        {
            nodo *q=raiz;
            if(q->der==NULL)
            {
                raiz=q->izq;cen=1;
            }
            else if(q->izq==NULL)
            {
                raiz=q->der;cen=1;
            }
            else
            {
                borrar(q->izq,q,cen);
                reestructura1(raiz,cen);
            }
            delete(q);
        }
    }
    else   { printf("ARBOL: No se encuentra");}
}

void nodo::reestructura1(nodo *&raiz,int&cen)
{
    if(cen==1)
    {
        switch(raiz->fe)
        {
            case -1:raiz->fe=0;break;
            case 0:raiz->fe=1;cen=0;break;
            case 1:
                nodo *raiz1=raiz->der;
                if(raiz1->fe>=0)
                {
                    raiz->der=raiz1->izq;
                    raiz1->izq=raiz;
                    switch(raiz1->fe)
                    {
                    case 0:raiz->fe=1;raiz1->fe=-1;cen=0;break;
                    case 1:raiz->fe=0;raiz1->fe=0;break;
                    }
                    raiz=raiz1;
                }
                else
                {
                    nodo *raiz2=raiz1->izq;
                    raiz->der=raiz2->izq;
                    raiz2->izq=raiz;
                    raiz1->izq=raiz2->der;
                    raiz2->der=raiz1;
                    if(raiz2->fe==1)  raiz->fe=-1;
                    else   raiz->fe=0;
                    if(raiz2->fe==-1) raiz1->fe=1;
                    else   raiz1->fe=0;
                    raiz=raiz2;
                    raiz2->fe=0;
                }
                break;
        }
    }
```

```
}

void nodo::reestructura2(nodo *&raiz,int&cen)
{
    if(cen==1)
    {
        switch(raiz->fe)
        {
            case 1:raiz->fe=0;break;
            case 0:raiz->fe=-1;cen=0;break;
            case -1:
                nodo *raiz1=raiz->izq;
                if(raiz1->fe<=0)
                {
                    raiz->izq=raiz1->der;
                    raiz1->der=raiz;
                    switch(raiz1->fe)
                    {
                        case 0:raiz->fe=-1;raiz1->fe=1;cen=0;break;
                        case -1:raiz->fe=0;raiz1->fe=0;break;
                    }
                    raiz=raiz1;
                }
                else
                {
                    nodo *raiz2=raiz1->der;
                    raiz->izq=raiz2->der;
                    raiz2->der=raiz;
                    raiz1->der=raiz2->izq;
                    raiz2->izq=raiz1;
                    if(raiz2->fe==-1) raiz->fe=1;
                    else   raiz->fe=0;
                    if(raiz2->fe==1)  raiz1->fe=-1;
                    else   raiz1->fe=0;
                    raiz=raiz2;
                    raiz2->fe=0;
                }
            break;
        }
    }
}

void nodo::borrar(nodo *&aux,nodo *&q,int&cen)
{
    if(aux->der!=NULL)
    {
        borrar(aux->der,q,cen);
        reestructura2(aux,cen);
    }
    else
    {
        q->codi=aux->codi;
        q=aux;
        aux=aux->izq;
        cen=1;
    }
}

void nodo::preorden(nodo *raiz)
{
    if(raiz!=NULL)
    {
        cout<<"  "<<raiz->codi;printf("%2d",raiz->fe);
        preorden(raiz->izq);
```

```
        preorden(raiz->der);
    }
}

void nodo::inorden(nodo *raiz)
{
    if(raiz!=NULL)
    {
        inorden(raiz->izq);
      mygx(11); cout<<raiz->codi;
      mygx(23);
           if(int(raiz->canal/10)==0) cout<<"0"<<raiz->canal;
           else cout<<raiz->canal;
       gotoxy(59,wherey());
      cout<<endl;
       inorden(raiz->der);
    }
}


void nodo::postorden(nodo *raiz)
{
    if(raiz!=NULL)
    {
        postorden(raiz->izq);
        postorden(raiz->der);
        cout<<"  "<<raiz->codi;printf("%2d",raiz->fe);
    }
}

int nodo::busca_avl(nodo *raiz,int clave)
{
    comp++;
    if(raiz!=NULL)
    {
        //comp++;
        if(clave<raiz->codi)
        return busca_avl(raiz->izq,clave);
        else
       {
        //comp++;
          if(clave>raiz->codi)
            return busca_avl(raiz->der,clave);
            else
            return 1;
       }
    }
    else
    return 0;//no se encuentra
}
int busca_array(int clave)
{
    int i=0;
    int cen=0;

    while (i<=tope && cen==0)
    {
        comp2++;
        if(clave==array[i].codi)
        cen=1;
        else
        i++;
    }
    if (cen==0) return 0;
```

```
    else { return 1;}}

nodo * nodo::nuevo_nodo(struct alumno clave)
{
    nodo *raiz=new(nodo);
    raiz->codi=clave.codi;
    raiz->canal=clave.canal;
    raiz->izq=NULL;raiz->der=NULL;raiz->fe=0;
    return raiz;
}

int nodo::altura(nodo *raiz)
{
    if(raiz!=NULL)
    {
        if(altura(raiz->izq)>altura(raiz->der))    return 1+altura(raiz->izq);
        else   return 1+altura(raiz->der);
    }
    else return 0;
}

int nodo::nodos(nodo *raiz)
{
    if(raiz!=NULL)return 1+nodos(raiz->izq)+nodos(raiz->der);
    else   return 0;
}

void nodo::muestra_nivel(nodo *raiz,int n)
{
    if(raiz!=NULL)
    {
        if(n==1)   cout<<raiz->codi<<" ";
        else
        {
            muestra_nivel(raiz->izq,n-1);
            muestra_nivel(raiz->der,n-1);
        }
    }
}

int nodo::nivel(nodo *raiz,int clave)
{
    if(raiz!=NULL)
    {

        if(clave<raiz->codi)  return 1+nivel(raiz->izq,clave);
        else if(clave>raiz->codi)    return 1+nivel(raiz->der,clave);
        else   return 1;
    }
    else   return 0;
}

//numero de nodos terminales u hojas
void nodo::peso(nodo *raiz,int &k)
{
    if(raiz!=NULL)
    if(raiz->izq==NULL&&raiz->der==NULL)   k++;
    else
    {
        peso(raiz->izq,k);
        peso(raiz->der,k);
    }
}
```

```
void mygx (int posf)
{
   int x;
   x=posf-wherex();
   for (int k=1; k<=x; k++)
   cout<<" ";
}
```