ESTRUCTURA DE DATOS AVANZADAS

Clases Disjuntas
Montículos
Árboles de Búsqueda

Autor: Miguel Ángel Bañolas Adrogué
Director: Xavier Franch Gutiérrez

Universitat Oberta de Catalunya (UOC)

Junio 2002

INDICE

1. Introducción

En los estudios de Ingeniería Técnica de Informática de Gestión de la UOC, se estudian estructuras básicas de datos y los algoritmos correspondientes.

En un trabajo previo realizado por Esteve Mariné, y dirigido por el director de este trabajo, se han implementado dichas estructuras creando una jerarquía de clases en leguaje Java.

En el presente trabajo, se amplían las estructuras con otras más avanzadas, y como veremos, más eficientes, integrándose en las jerarquías del anterior trabajo.

2. Objetivos

Los objetivos de este trabajo son

- Crear una interficie para las nuevas estructuras de datos integrada en la jerarquía de tipos abstractos de datos (TAD) ya existentes.

- Implementar en java la estructura de datos, siguiendo los principios metodológicos de orientación a objetos.

- Diseñar y ejecutar unas pruebas, para probar el correcto funciona-miento y su eficiencia en términos absolutos.

- Implementar un visualizador sencillo que muestre la evolución de la estructura cuando se efectúen operaciones correspondientes al TAD.

Las estructuras seleccionadas son:

- Clases Disjuntas (MFSets), utilizando los algoritmos de nodos con encadenamientos y con estructura de árbol.

- Montículos (Heaps), con algoritmo Binario, Binomial y Fibonacci.

- Árbol Binario de búsqueda utilizando el algoritmo denominado Árbol Red-Black (Red-Black Tree).

La eficiencia relativa se estudia entre las estructuras del mismo tipo entre sí y, en caso del árbol Red-Black, se compara con los TADs Árbol de búsqueda encadenado y Árbol AVL, implementados en el trabajo realizado por Esteve Mariné,

3. Descripción Estructuras

3.1. Clases disjuntas

Descripción

La estructura de clases disjuntas (Disjoint Sets) gestiona colecciones de datos (clases), identificando cada colección por un identificador, que puede ser cualquier elemento de la colección.
El identificador de una clase siempre será el mismo, si no se ha modificado la clase.

Operaciones

Las operaciones comunes de las clases disjuntas son:

novaClasse(Comparable id, Object elem)

Crea una nueva clase cuyo único miembro es el elemento x. El identificador de la clase es el propio elemento x.
Se requiere que x no exista en la estructura.

Comparable unio(Comparable id1, Comparable id2)

Une en una única clase la clase que contiene x y la clase que contiene y, creando una nueva clase.
Los elementos x, y deben ser disjuntos, pertenecientes a dos clases distintas.
El identificador de la nueva clase, puede ser cualquiera de los identificadores de las clases unidas.
Las clases originales que se han unido, se destruyen (no puede existir un mismo elemento en dos clases diferentes.)

Comparable trobaClasse(Object elem)

Devuelve el identificador de la clase que contiene el elemento x.

boolean congruents(Object elem1, Object elem2)

Devuelve verdadero o falso, según los elementos x e y pertenecen a una misma clase o no.

Algoritmos

Se han aplicado dos algoritmos distintos descritos en el libro Cormen [1].

Encadenado

Características

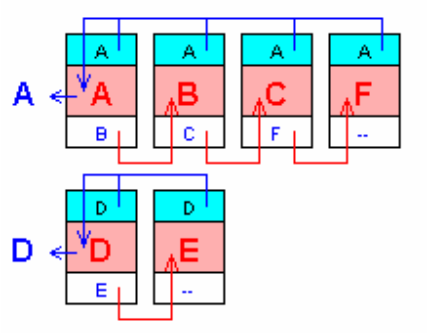
Cada clase esta formada por:

- una lista con doble encadenamiento conteniendo los elementos. Los encadenamientos son hacia el siguiente elemento de la lista y hacia el elemento representante de la clase.

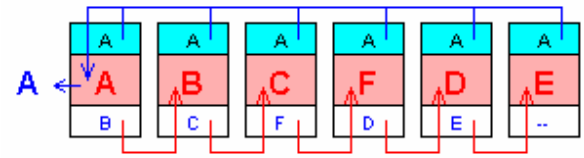
- un puntero que apunta al último elemento de la lista.

El identificador de la clase es el primer elemento de la lista.

Es de destacar que el puntero que señala el último elemento no es absolutamente necesario, pero el coste de mantenimiento es pequeño y facilita la operación de unión (no requiere recorrer toda la lista para encontrar el último elemento).

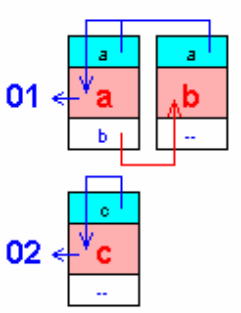


Unión

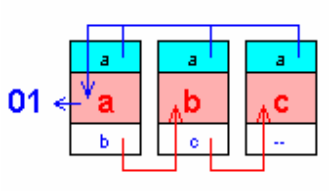


Representación de dos clases, cuyos identificadores son A y D, y su unión (representante A).

La implementación se ha realizado considerando sólo elementos, cuyo identificador es el propio elemento (según se muestra en la figura anterior) y elementos asociados a un identificador, utilizando la clase ClauValor que relaciona un valor y una clave.



unión



Implementación con objetos valor-identificador.

La estructura se completa gestionando las distintas clases mediante una estructura de datos que permita la búsqueda de elementos.

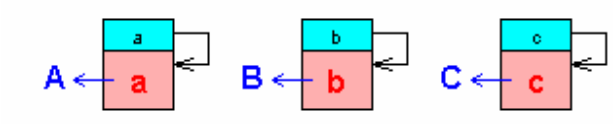
El algoritmo utilizado corresponde a la variante heurística que selecciona la clase de menor número de elementos para colocar a la cola de la de mayor número de elementos, para así tener que actualizar un número menor de encadenamientos. El número de elementos de la lista se almacena y actualiza en el nodo identificador de la clase.

Árboles

Características

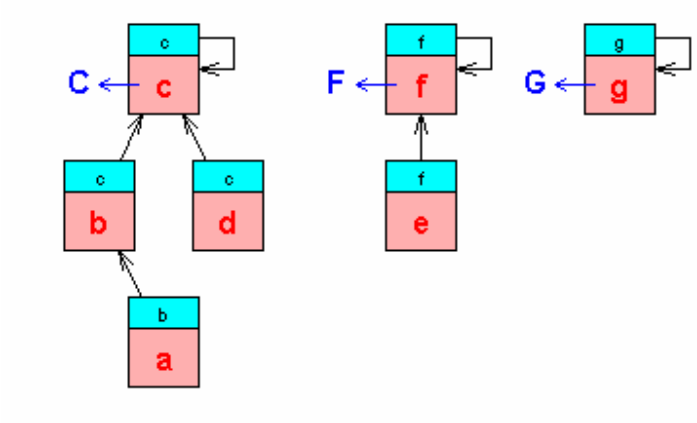
Cada clase se representa por un árbol constituido por nodos que tienen un único encadenamiento a su padre.

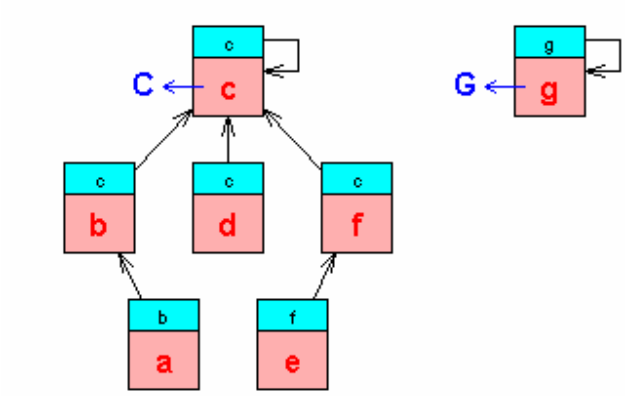
Cuando se crea una clase, se genera un árbol con un único elemento. El encadenamiento apunta a sí mismo.



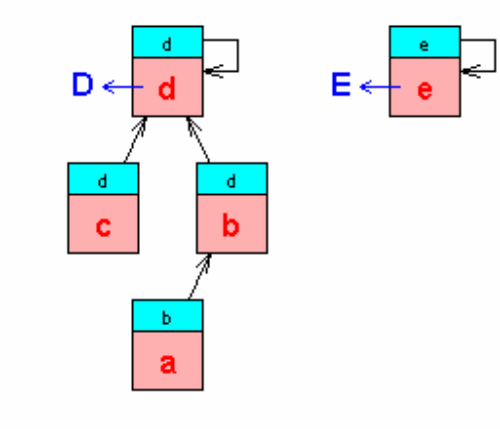
Conjunto de tres clases con un elemento cada una de ellas a, b, c. Sus identificadores son A, B, C.

En la unión, el nodo identificador de la clase que tenga menor número de elementos, cambiará su encaminamiento al nodo de la otra clase, y cambiará los encadenamientos al nodo identificador de la nueva clase.
Para ello, se mantiene un parámetro en los nodos que indica el 'ranking' del nodo (número de nodos que tiene por debajo). De esta forma para unir dos árboles (clases), el nodo raíz (identificador) con menor ranking tomará como padre al nodo raíz del árbol con mayor ranking.

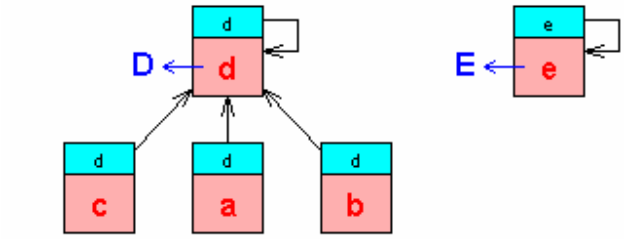




La figura muestra un ejemplo de unión. La raíz del árbol con identificador F apunta hacia la raíz del árbol con mayor número de elementos.
También se aplica otra heurística por la en la operación de trobaClasse, todos los nodos consultados son dirigidos hacia la raíz, tal como se muestra en el ejemplo, por lo que el camino hacia el nodo identificador de la clase se acorta. En esta operación no cambia el parámetro ranking, ya que el número de elementos de la clase permanece invariable.



Conjunto con dos clases: D y E



Mismo conjunto después de haber efectuado trobaClasse(a)

3.2. Montículos

Descripción

Los montículos (Heaps) son estructuras de datos que permite la extracción de los mismos de forma ordenada. Existen dos tipos: los montículos máximos, que permite extraer los elementos en orden decreciente y los montículos mínimos, en que la extracción se inicia por el más pequeño y termina con el valor máximo.

En este trabajo trataremos únicamente los montículos mínimos.

Operaciones

Las operaciones comunes de las clases disjuntas son:

crea()

Crea un montículo vacío sin ningún elemento.

afegeix(Comparable clauElem, Object valorElem)

Añade un elemento al montículo con una determinada prioridad que debe tener elemento insertado.

ClauValor minimo()

Devuelve el valor mínimo del montículo. El elemento permanece en el montículo, por lo que mientras no se modifique el montículo, el elemento mínimo siempre será el mismo.

ClauValor extraeMinimo();

Elimina el elemento con prioridad mínima del montículo.

disminuir(Elemento x, Prioridad p)

Asigna una nueva prioridad al elemento x. Esta prioridad debe ser inferior a la actual.

Comparable buscar(Object elemento)

Devuelve la prioridad del elemento.

boolean existe(Object elemento)

Devuelve verdadero o falso, según exista o no el elemento en el montículo.

boolean buit()

Devuelve verdadero si el montículo está vacío o falso si contiene algún elemento

int nbElems()

Devuelve le número de elementos del montículo.

copia(Montículo m)

Crea un montículo con los elementos de otro.

También, a efectos de la visualización grafica, se ha dotado de la operación

esborra(Object elemento)

Que permite eliminar un elemento del montículo.

Algoritmos

Se han aplicado tres algoritmos distintos descritos en el libro Cormen [1].

Binario

Características

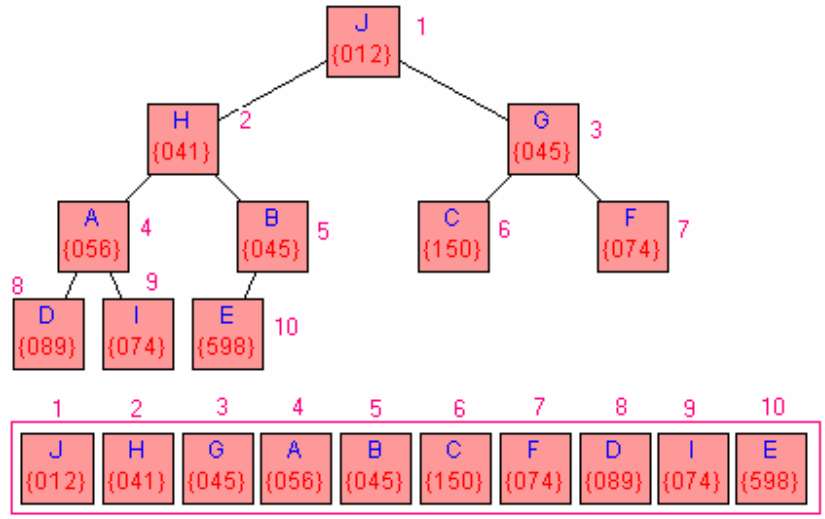
Consiste en un vector en el que se van incorporando los elementos del montículo de tal forma que:

- el vector está lleno desde la posición 1 hasta la posición n, siendo n el número de elementos del montículo, sin dejar, por tanto, ninguna posición vacía.

- para todo elemento del vector en posición i, los elementos en posiciones 2*i y 2*i+1, tienen prioridad menor que el elemento en i.

Su representación en forma de árbol consiste en un árbol binario, en las que los elementos hijos de un elemento i, son los elementos 2*i, izquierda y 2*i+1, derecha. El padre de cada nodo siempre tendrá una prioridad igual o menor que la del hijo.
El árbol tiene completo cada nivel, a excepción del último que puede estar parcialmente lleno, pero completo de izquierda a derecha hasta llegar al último.

La raíz del árbol es el elemento mínimo.



Ejemplo de montículo binario.

Para mantener la estructura con estos requisitos se recurre al método heapify, una vez introducido el elemento el vector (árbol) hace que éste ‘reflote’ hasta alcanzar su posición adecuada.

Binomial

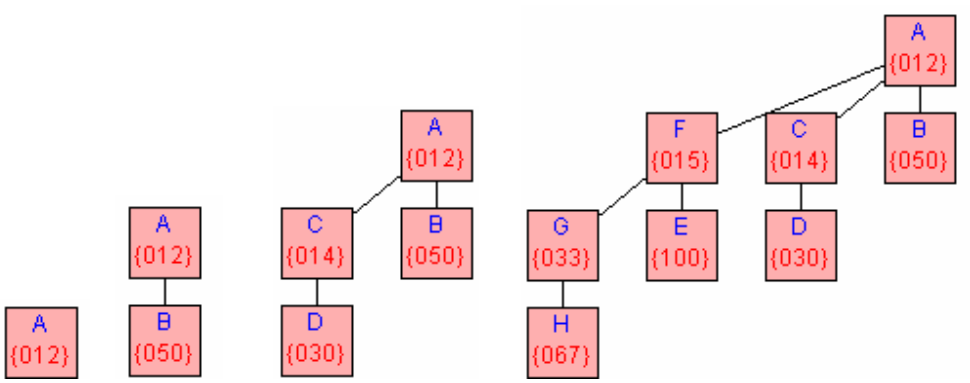
Características

Un montículo Binomial es una colección de árboles binomiales, los cuales se definen recursivamente de la siguiente forma:

- El árbol B_0 es el que tiene un solo elemento.

- Un árbol B_k consiste en dos árboles B_{k-1} que están unidos juntos, siendo la raíz de uno, el hijo más a la izquierda de la raíz del otro.

Ejemplos:



B_1 B_2 B_3 B_4

Los árboles binomiales de un montículo Binomial deben tener las siguientes propiedades:

- La prioridad de un nodo siempre es superior a la de su padre (montículos mínimos), por lo que en cada subárbol la raíz tiene la prioridad mínima.

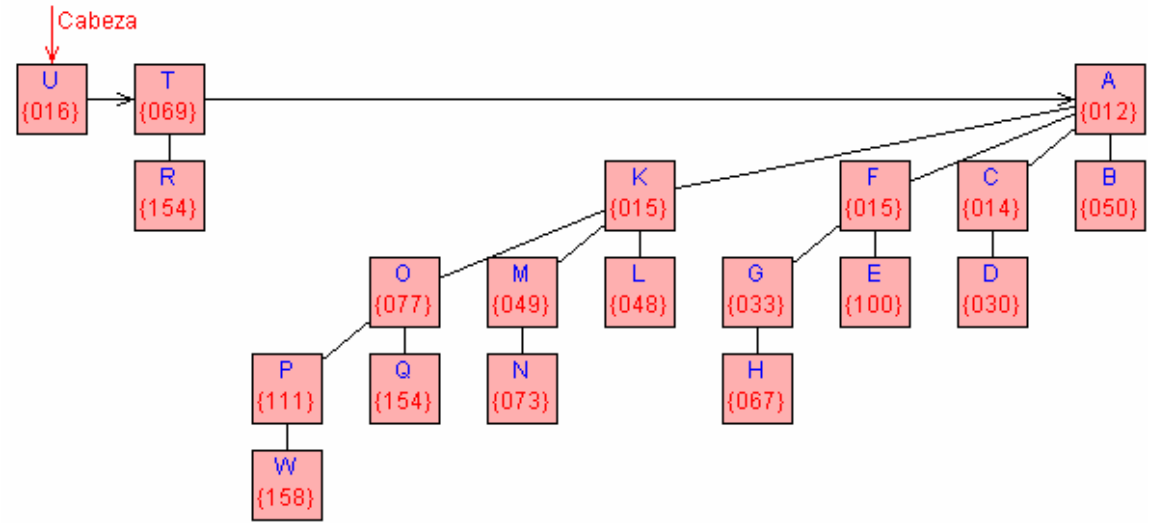
- Para todo valor positivo k, existe al menos un árbol Binomial en el montículo, cuya raíz tiene grado k.

Para utilizar esta estructura, cada nodo (con su prioridad y valor asociado) tiene tres encadenamientos: un encadenamiento al padre, otro al hijo situado más a la izquierda y otro al hermano situado inmediatamente a su derecha. Además tiene un atributo indicando su grado, que es el número de hijos del nodo.

El montículo tiene un parámetro que apunta al nodo inicial (cabeza).

El detalle de esta estructura, así como el mecanismo de funcionamiento de los algoritmos de las operaciones del montículo, se encuentran en las páginas 459 a 471 de la obra de Cormen [1].

Ejemplo de montículo Binomial:



Fibonacci

Características

También consiste en una colección de árboles. Los árboles no están ordenados como sucede en el caso de los montículos binomiales, pero si están enlazados las raíces.

Cada nodo contiene

- un encadenamiento al nodo padre:

- un encadenamiento al nodo de uno de sus hijos.

- un encadenamiento circular a sus hermanos hacia la derecha.

- un encadenamiento circular a sus hermanos hacia la izquierda.

En el caso que el nodo no tenga hermanos se encadena hacia sí mismo.

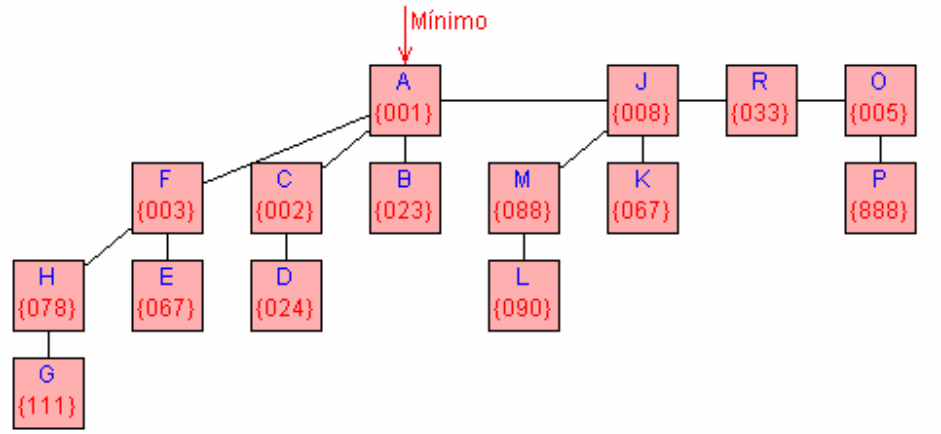
Además cada nodo tiene dos parámetros:

- el número de hijos de la lista de hijos

- y una marca indicando si un nodo determinado ha perdido un hijo desde la última vez que fue asignado hijo de otro nodo.

Remito a la obra de Cormen [1] para tener detalle gráfico de la estructura, así como explicación del mecanismo de los algoritmos.

Ejemplo de montículo de Fibonacci:



Los tiempos necesarios para las operaciones en montículos, conteniendo *n* elementos, se resumen en la siguiente tabla

Operaciones	Montículo Binario (Más desfavorable)	Montículo Binomial (Más desfavorable)	Montículo Fibonacci (amortizado)
crear	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
afegeix	$\Theta(\log n)$	$O(\log n)$	$\Theta(1)$
minimo	$\Theta(1)$	$O(\log n)$	$\Theta(1)$
extraeMinimo	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$
unio	$\Theta(n)$	$O(\log n)$	$\Theta(1)$
disminur	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
borrar	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$

Como se puede observar, el algoritmo más eficiente es el de Fibonacci que tiene un coste de O(1), en todas las operaciones excepto en aquellas en las que es necesario eliminar un elemento, donde la eficacia es, como máximo, igual que la de los otros algoritmos.

Destacar también que el algoritmo binario es igual o más eficiente que el Binomial excepto en la operación unio.

3.3. Árboles Binarios de Búsqueda

Descripción

Los Árboles Binarios de Búsqueda son estructuras en forma de árbol donde cada elemento puede tener uno o dos ramas (hijos) y, a excepción del primer elemento (raíz) un padre.

Estas estructuras de datos están dotadas de operaciones dinámicas tales como la inserción, consulta y eliminación.

Operaciones

Las operaciones comunes de las clases disjuntas son:

crea()

Crea un árbol vacío.

afegeix(Comparable clauElem, Object valorElem)

Añade un elemento asociándoles la clave indicada. Esta clave debe ser un objeto Comparable.

esborra(Comparable clauElem)

Borra el elemento correspondiente a la clave.

ClauValor consulta(Comparable clauElem)

Busca y devuelve el elemento correspondiente a la clave. El elemento permanece en la estructura.

boolean existeix(Comparable clauElem)

Comprueba si existe un elemento o no, devolviendo cierto o falso respectivamente.

posiciona()

Posiciona el cursor en la primera posición de los elementos ordenados en inordre.

Object consulta()

Devuelve el elemento indicado por el cursor.

avanca()

Avanza el cursor una posición, según la ordenación inordre. En el caso de que esté al final, no mueve el cursor.

boolean fi()

Comprueba si ha llegado al final, devolviendo verdadero o falso.

Enumeration elements()

Devuelve un objeto Enumeration conteniendo los elementos ordenados en inordre.

boolean buit()

Devuelve verdadero o falso según la estructura esté vacía o contenga algún elemento.

int nbElems()

Devuelve el número de elementos de la estructura.

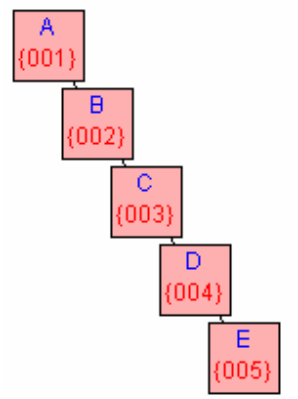
Algoritmos

Se han aplicado tres algoritmos distintos, los dos primeros Árbol Binario Encadenado y Árbol Binario AVL, implementados por Esteve Mariné y un tercero, denominado Árbol Red-Black, siguiendo las directrices de la obra de Cormen [1]
Árbol Binario Encadenado

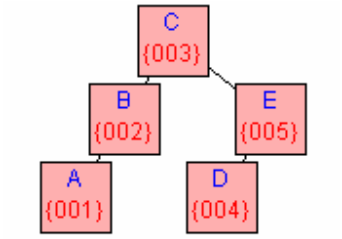
El algoritmo construye un árbol binario caracterizado por que la raíz es mayor que todos los elementos del subárbol izquierdo y más pequeño que todos los elementos del subárbol derecho. Los subárboles derecho e izquierdo cumplen, asimismo, con estas características.

La estructura del árbol y el número de subárboles depende de la secuencia de introducción.

Se denomina encadenado debido a que cada nodo contiene un encadenamiento a los nodos hojas izquierda o derecha si existen,



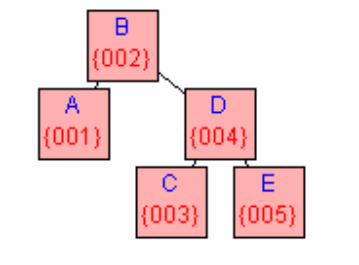
Árbol binario construido con
La secuencia de entrada
A, B, C, D, E



Árbol binario construido con
La secuencia de entrada
C,B,A, E, D

Árbol Binario AVL

Cumple con las características de un Árbol Binario de búsqueda descrito anteriormente y adicionalmente, para cada nodo la altura de los subárboles izquierdo y derecho difieren como mucho en una unidad. Es decir tienen la altura equilibrada.



Árbol binario AVL construido con
La secuencia de entrada
A, B, C, D, E

Árbol Binario Red-Black

Cumple con las características de un Árbol Binario de Búsqueda, pero además cada nodo tiene un atributo adicional que es su color, el cual puede ser rojo o negro y un encadenamiento hacia el nodo padre.

La construcción de un árbol Red-Black asegura que ningún camino desde la raíz hasta cualquiera de las hojas extremas es mayor que el doble de cualquier otro. El árbol esta parcialmente equilibrado.

Sus características son:

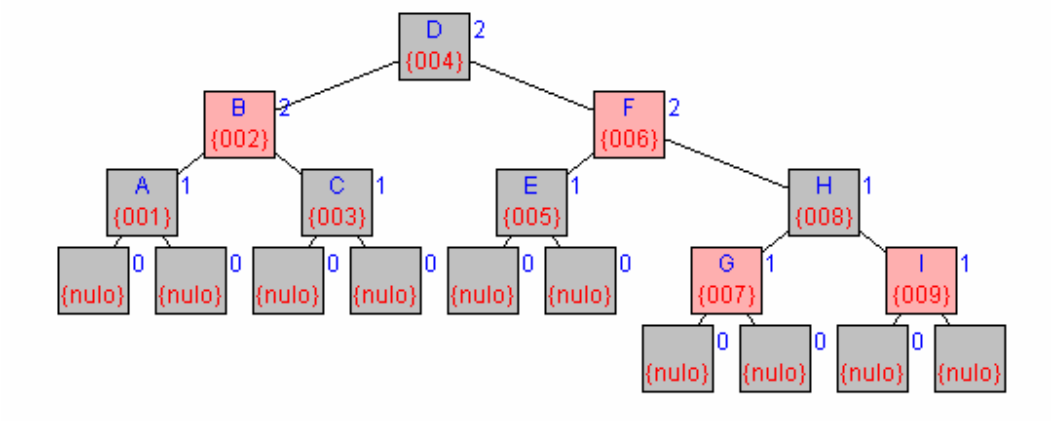
- Cada nodo tiene un color que puede ser rojo o negro.

- La raíz del árbol es negra.

- Todas las hojas (nodos extremos de un camino) son negras.

- Si un nodo es rojo, sus dos hijos son negros.

- Para cada nodo, sus caminos descendientes desde el nodo hasta las hojas extremas, contienen el mismo número de nodos negros. Este número se denomina altura negra del nodo.



Ejemplo de árbol Red-Black. En la parte superior derecha de cada nodo se indica su altura negra.

Es de señalar la necesidad de un nodo negro 'fantasma' con clave nula al que apunta los encadenamientos hijos izquierdo y derecho de los nodos hojas extremos.

4. Programa y visualizadores

4.1. Instalación

Para utilizar el programa se requiere tener instalado correctamente la aplicación Java jkd 1.2.1.

Concretamente las pruebas las he realizado en un PC Pentium III con 256 k de memoria. La aplicación java está instalada en el directorio: C:\jdk1.2.1,
y los comandos en autoexec.bat son:

```
set path=%PATH%;c:\JDK1.2.1\BIN;
set CLASSPATH = .;
```

No he podido comprobar los requerimientos mínimos de memoria, pero sí que he comprobado la lentitud de la aplicación cuando se tienen otras aplicaciones abiertas.

Para la instalación del programa se debe descomprimir el archivo adjunto, TADsAv.zip, en un directorio determinado, por ejemplo TADsAv y ejecutar el programa compile.bat, ya sea desde el Sistema Operativo MDOS o desde Windows:
Inicio -> Ejecutar -> c:\TADsAv\compile.bat

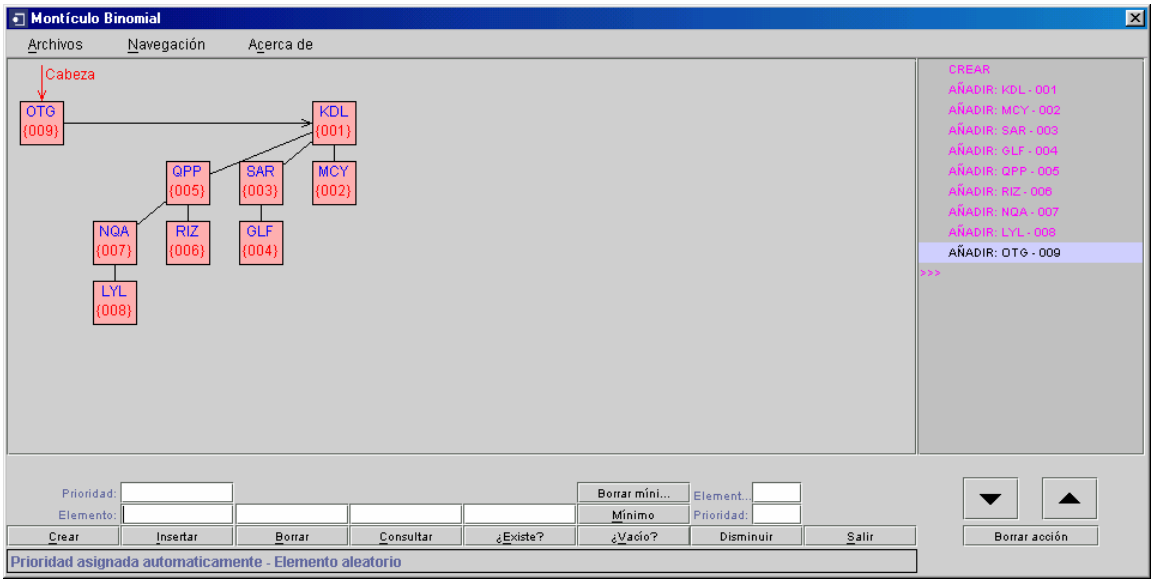
Una vez compilado, se inicia automáticamente la ejecución de Inicio y se muestra la ventana de entrada. Seleccionando uno de los botones y pulsando Run, o con un doble clic del ratón, se inicia una aplicación concreta.

Se han creado dos tipos de visualización:

4.2. Visualizadores

4.2.1. Visualización evolución de las estructuras

Para cada una de las estructuras tratadas, se muestra la formación de la misma y las operaciones descritas.



Ejemplo del visulaizador de la evolucion de una estructura Montículo Binomial.

El funcionamiento general es el siguiente:

Crear una estructura, lo que se puede efectuar de varias formas:

- crear una estructura vacía: pulsando el botón *Crear*. Asegurarse que el cuadro de texto *Elemento* esté vacío, de lo contrario efectuaría la siguiente opción.

- crear una estructura con un número determinado de elementos: indicar los elementos en el cuadro de texto *Elemento* y pulsar crear o en el menú *Navegación -> Generar Estructura*

- crear una estructura desde un archivo: *Archivo -> Abrir.* El archivo debe contener una estructura del mismo tipo.

- Si, una vez generada una estructura, se vuelve a pulsar *Crear*, se borra la actual y se inicia una nueva.

Efectuar una determinada operación.

- introducir los datos correspondientes a la operación y pulsar el botón de la misma.

- en la operación añadir, si no se introduce el elemento y/o la prioridad/clave, esta se genera automáticamente.

- los elementos pueden ser tres caracteres alfanuméricos. Si se introduce más, se ignoran.

- las prioridades o claves deben ser como máximo tres dígitos. La aplicación introduce ceros en la izquierda para completar la longitud de 3 dígitos.

- si la operación no se puede efectuar, se genera un mensaje.

Lista de acciones:

- las operaciones que modifican la estructura se registran en la lista de la derecha, que es útil para

a) conocer lo que se ha realizado.

b) hacer evolucionar hacia atrás y hacia delante las operaciones realizadas. Se puede realizar o bien con los botones con flechas al pie de la lista o simplemente pulsando una de las operaciones realizadas. La estructura se genera de nuevo hasta la operación indicada. El símbolo >> señala la siguiente operación a realizar.

c) deshacer una operación, mediante el botón *Borrar acción.*

d) insertar una operación. Se debe generar la estructura hasta el punto donde se debe insertar (según lo indicado en b)) y realizar la operación correspondiente. Las sucesivas operaciones registradas se mantienen.

Grabar y leer una estructura.

Las estructuras generadas, se pueden grabar para después ser leídas. Se graba toda la secuencia de operaciones que aparece en la lista, independientemente que en pantalla se muestre hasta una determinada operación.

Las estructuras se graban con el sufijo ‘tad’ y se incorpora con los datos grabados el tipo de estructura (Árboles Búsqueda, Montículos, Clases Disjuntas), para poder después ser leídas por la correspondiente aplicación.

La forma de realizarlo es la típica de la mayoría de programas. (*Archivos -> Abrir, -> Guardar, -> Guardar como).*

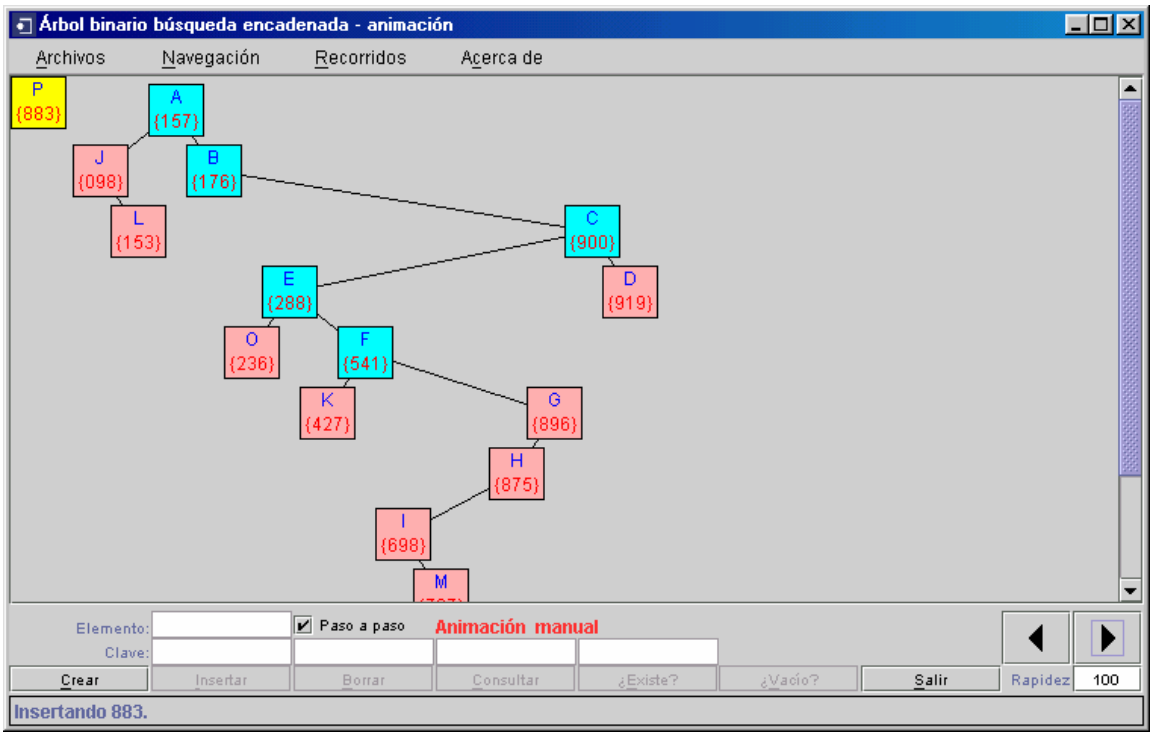
Visualización animada

Se ha incorporado también una visualización del árbol de búsqueda encadenada con una animación mostrando el camino que sigue la comparación de nodos hasta encontrar el lugar correspondiente, ya sea para añadirlo, borrarlo, consultarlo o ver si existe.

Este trabajo se ha realizado también bajo la dirección de Xavier Franch en una etapa previa a este trabajo con objeto de iniciar el estudio sobre la estructura de datos ya implementados y estudiar alternativas de visualización.

Su utilización es similar a la expuesta excepto el registro de las operaciones realizadas y su manipulación, que únicamente está implementado para realizarlo desde el menú.

En esta visualización se puede realizar de forma continua o bien de forma paso a paso, lo cual se consigue activado paso a paso.



Visualizador en su modo paso a paso. Muestra una etapa intermedia para añadir el elemento P con clave 883.

Otras

- En la estructura Árbol Red-Black, se ha incluido una casilla para poder mostrar o no, los nodos nulos hojas. De esta forma se consigue la posibilidad de tener una estructura mostrando sólo los nodos con elementos.

- Se ha incorporado también una visualización del árbol de búsqueda encadenada con una animación mostrando el camino que sigue la comparación de nodos hasta encontrar el lugar correspondiente, ya sea para añadirlo, borrarlo, consultarlo o ver si existe. Este trabajo se ha realizado también bajo la dirección de Xavier Franch en una etapa previa a este trabajo con objeto de iniciar el estudio sobre la estructura de datos ya implementados y estudiar alternativas de visualización.

4.2.2. Visualización evaluaciones de los TADs

Como se indicará más adelante se ha preparado un sistema para evaluar la eficiencia comparativa de los distintos algoritmos para un mismo tipo de estructura, en concreto para evaluar los árboles de búsqueda, los montículos y las clases disjuntas.

La visualización y opciones de estas evaluaciones son comunes para los tres tipos de estructura

Ejemplo de evaluación de montículos en la operación de inserción.

Para su utilización se debe:

- Introducir el número de elementos con los que se aplicará la operación. Si se deja en blanco se toma por defecto 2000.
- Introducir el número de repeticiones, por defecto tiene asignado 10. Las repeticiones se realizan siempre con la misma secuencia de datos. Tiene como objeto corregir los tiempos diferentes de ejecucion de aplicaciones que tiene Windows, dependiendo de la ejecución de otros programas.
- marcar las estructuras que se desean evaluar. Por defecto están marcadas todas las posibles. Esta opción se ha incluido para poder excluir algunas estructuras que difieran mucho del resto y, así, personalizar el estudio.
- Pulsar el botón de la operación que se desee evaluar.

Una vez terminado se muestra un gráfico de barras con indicación de los tiempos en milisegundos promedio (después de realizar un tratamiento estadístico de los mismos) y su intervalo con una probabilidad del 95 % (pequeña línea sobre el resultado.

Se debe tener presente la lentitud de algunas operaciones, por lo que es necesario seleccionar adecuadamente el número de elementos y/o repeticiones.

Este visualizador también tiene la opción de guardar los resultados, los cuales se pueden volver a consultar con la misma aplicación de evaluación. La grabación tiene en cuanta el tipo de estructura evaluada.

Los datos grabados, además de los básicos representados, también incluyen el detalle de cada repetición y el resultado del tratamiento estadístico. Estos datos se pueden consultar con cualquier programa capaz de leer archivo de texto, por ejemplo Word o, mayor, si se desean realizar cálculos Excel. En el apéndice se muestra la estructura interna del archivo grabado (en este caso leido con Excel).

4.3. Uso de TADs

Para la utilización de las estructuras simplemente se debe crear un objeto de la estructura con algún constructor de la clase y utilizar los métodos públicos de la misma.

Se incluye documentación generada por Java con la descripción de los contructores y métodos (subcarpeta doc).

Ejemplo:

```
HeapFibonacci h=new HeapFibonacci();
h.afegeix(new Integer(10), "A");
h.disminuir("A",new Integer(8));
```

Los TADs implementados en este trabajo, tienen los siguientes constructors (sin parámetro):

Árboles de búsqueda:

ArbreBinCercaEnc()
ArbreBinCercaRB()
ArbreBinAVL()

Montículos:

HeapBinario()
HeapBinomial()
HeapFibonacci()

Clases disjuntas:

MFSetEnc()
MFSetIdArbre()
MFSetIdEnc()

5. Evaluaciones

5.1. Descripción del cálculo

Se ha preparado la evaluación comparativa de las eficiencias de los distintos algoritmos, contabilizando el tiempo necesario para completar un número determinado de operaciones.

Todas las evaluaciones se realizan con una lista de datos (conjuntos de clave/prioridad y elemento asociado), cuyo número se puede fijar de antemano.
Se debe tener en cuanta que lista mus pequeñas consumen muy poco tiempo y su resultado es pequeño con una dispersión de los mismos muy elevada.

A partir de la lista generada, se general otras dos con los mismos datos, ordenadas de menor a mayor (creciente) y de mayor a menor (decreciente).

Las distintas listas generadas pueden representar el caso más favorable o desfavorable para algunas estructuras, pero para otras no es este el caso. No obstante, se ham seleccionado este tipo de listas comunes para tener una referencia comparativa.

Esta evaluación no pretende el determinar el orden de eficiencia de la estructura, sino la comparación de las mismas. Para determinar el orden de eficiencia se

tendría que realizar con un número mas elevado de datos y analizar los resultados en función del número de elementos.

Nos obstante, aparte del estudio realizado, sí es posible observar la dependencia existente entre el número de operaciones y los tiempos necesarios.

Los tiempos se registran para cada conjunto de operaciones realizadas, por ejemplo si se analiza la inserción de 10000 elementos, el tiempo registrado es el tiempo necesario para realizar las 10000 inserciones. Los procesos adicionales no se contabilizan en este tiempo.

Las repeticiones se realizan a partir de la misma lista.

Los resultados de las distintas repeticiones se analizan estadísticamente, primero descartando los valores que según la dispersión del conjunto. Se considera un valor no válido aquel que esta fuera del rango:

(Valor medio) $\pm$ C

siendo σ la desviación estándar y t el valor de t de Student con una probabilidad del 0.05 para N= numero de valores –1.

Con los valores válidos se calcula el nuevo promedio de las repeticiones y se calculas el intervalo de los mismos con una probabilidad del 95 %.

Intervalo = σ_m t/raíz (número de elementos).

Los resultados de las repeticiones, los valores rechazados y los resultados estadísticos se pueden conocer leyendo el archivo grabado después de efectuar el cálculo.

5.2. Resultados

En las siguientes páginas se exponen los resultados para cada una de la evaluaciones.

A destacar que para algunas de ellas, primero se exponen la comparación de todas, y como una de ellas se excede respecto a las otras, el siguiente análisis se realiza excluyendo la que requiere más tiempo.

Los comentarios de los resultados, se indicarán en la presentación de los mismos.

6. Conclusiones

En el presente trabajo se han implementado con éxito los algoritmos utilizando Java.

Se ha presentado gráficamente la evolución de las estructuras pudiendo hacer y deshacer operaciones para observar como se ha realizado.

Además, se ha generado un medio de comparación de los tiempos necesarios para cada algoritmo y la posibilidad de realizar simulaciones variando el número de elementos y repeticiones.

7. Trabajos posteriores

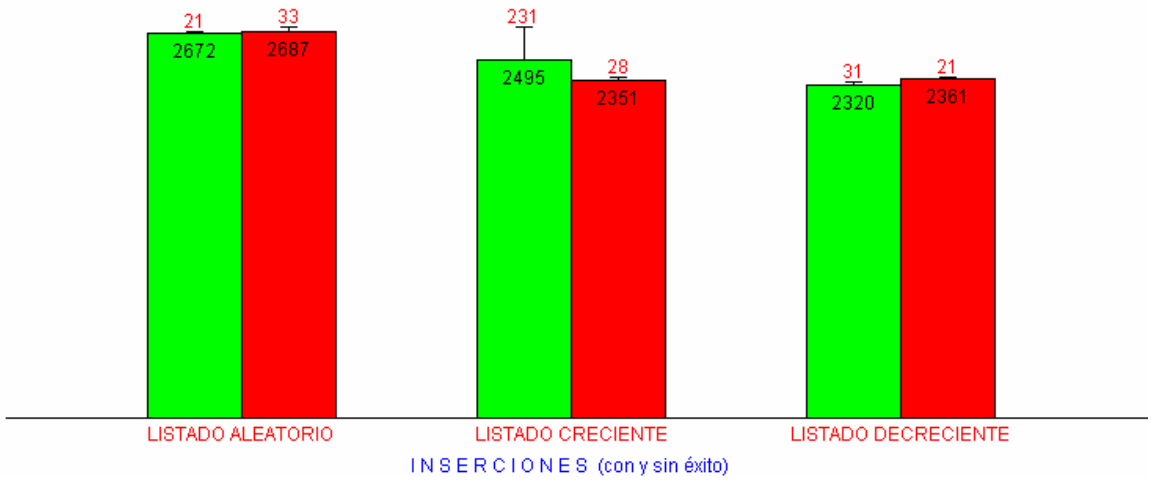
Los TADs generados, se pueden considerar como terminados (excepto si se encuentra algún error).

Es evidente que se puede profundizar más en las evaluaciones, teniendo mayor 'libertad' en la generación de las listas y o operaciones realizadas.

Sobre las visualizaciones de las evoluciones, sería interesante el poder ver dinamicamente como actúan los distintos algoritmos, mostrando los cambios de la estructura, por ejemplo, en las rotaciones y otras operaciones particulares.

También sería interesante el poder incluir otras TADs en las visualizaciones. Para ello me comprometo a describir el procediendo seguido para realizar las mismas en una entrega posterior (la falta de tiempo me impide entregarlo ahora, así como explicar mejor los resultados obtenidos en las evaluaciones).

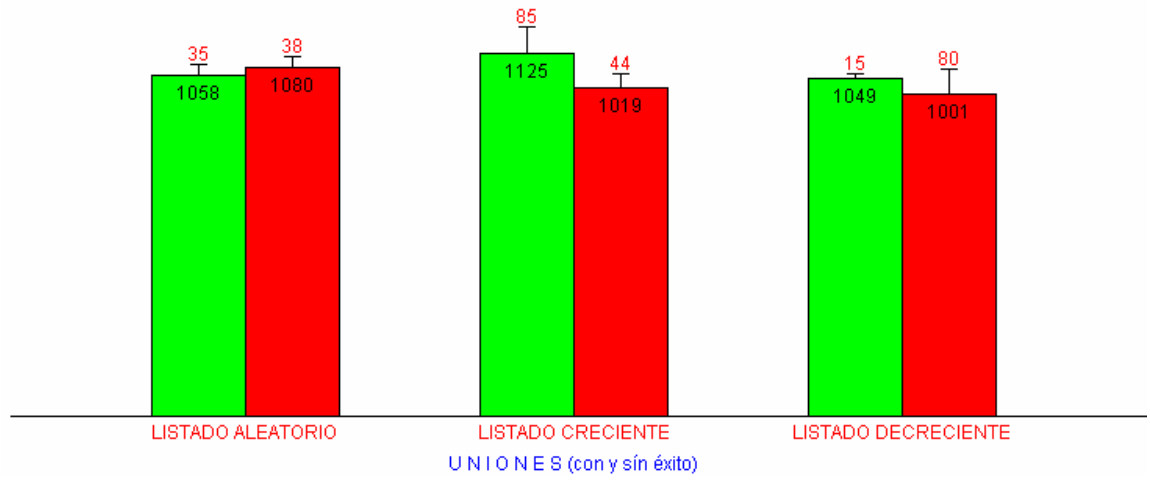
Clases Disjuntas
Operación de creación de Clases con un elemento



Encadenado Árbol

Clases Disjuntas	Operación novaClasse		
Núm. Elementos:	10000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Algoritmo encadenado	2672 ± 21	2495 ± 231	2320 ± 31
	2714 ± 184	2339 ± 21	2332 ± 22
	2747 ± 196	2500 ± 229	2372 ± 17
	2702 ± 30	2351 ± 18	2361 ± 19
	2727 ± 21	2361 ± 3	2362 ± 20
Promedio	**2712 ± 90**	**2409 ± 100**	**2349 ± 22**
Algoritmo Árbol	2687 ± 33	2351 ± 28	2361 ± 21
	2661 ± 34	2351 ± 18	2362 ± 29
	2673 ± 35	2363 ± 21	2362 ± 20
	2836 ± 36	2351 ± 34	2363 ± 30
	2774 ± 37	2357 ± 25	2343 ± 29
Promedio	**2726 ± 35**	**2355 ± 25**	**2358 ± 26**

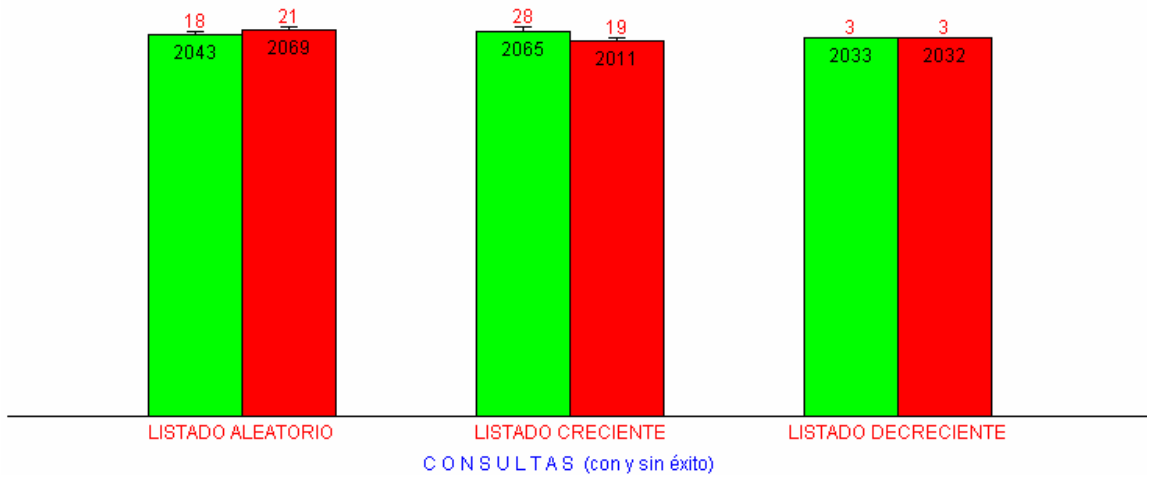
Clases Disjuntas
Operación de Unión de clases



Encadenado Árbol

Clases Disjuntas	Operación unio (con y sin éxito)		
Núm. Elementos:	10000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Algoritmo encadenado	1058 ± 35	1125 ± 85	1049 ± 15
	1050 ± 15	1013 ± 57	1160 ± 71
	1043 ± 29	1031 ± 20	1001 ± 17
	1068 ± 20	1013 ± 21	1024 ± 29
	1043 ± 21	1043 ± 98	1037 ± 17
Promedio	**1052 ± 24**	**1045 ± 56**	**1054 ± 30**
Algoritmo Árbol	1080 ± 33	1019 ± 44	1001 ± 80
	1070 ± 34	1105 ± 67	996 ± 16
	994 ± 35	933 ± 4	958 ± 22
	994 ± 36	970 ± 23	988 ± 30
	1014 ± 37	947 ± 19	964 ± 23
Promedio	**1030 ± 35**	**995 ± 31**	**981 ± 34**

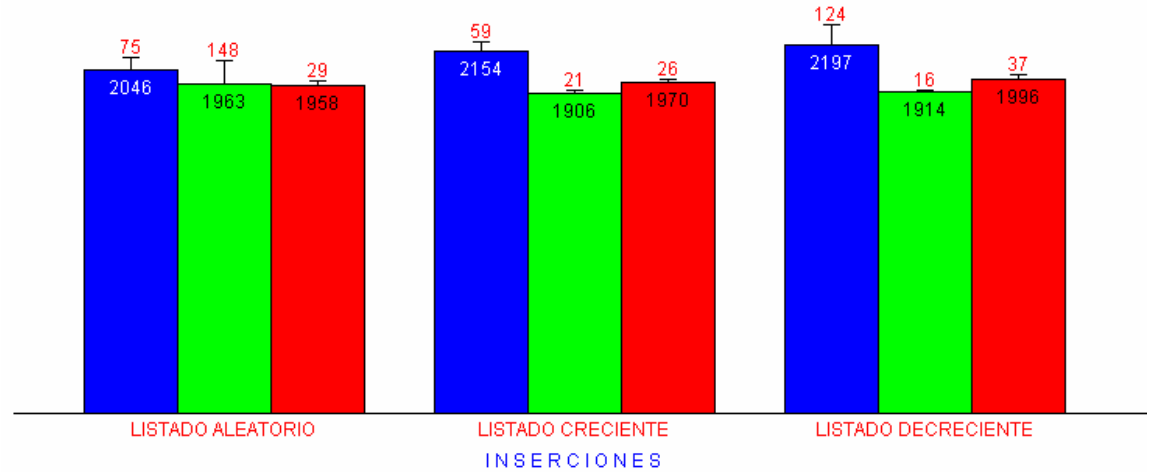
Clases Disjuntas
Operación de Consultas de Clases



Encadenado Árbol

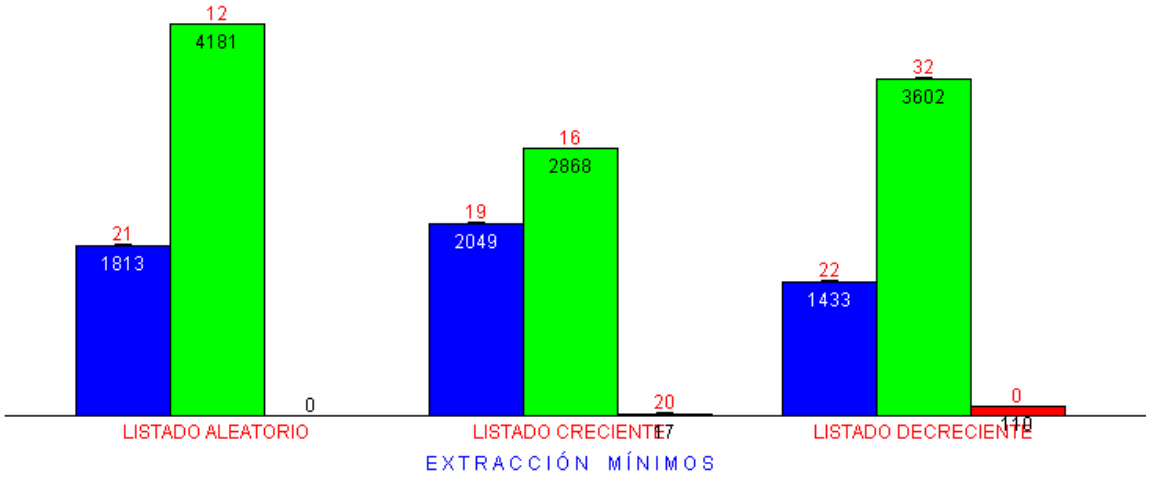
Clases Disjuntas	Operación trobaClasse (con y sin éxito)		
Núm. Elementos:	10000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Algoritmo encadenado	2043 ± 18	2065 ± 28	2033 ± 3
	1978 ± 17	2011 ± 28	2011 ± 19
	1960 ± 23	2016 ± 18	2063 ± 111
	2044 ± 20	2076 ± 16	2027 ± 25
	2027 ± 22	2022 ± 16	2009 ± 20
Promedio	**2010 ± 20**	**2038 ± 21**	**2029 ± 36**
Algoritmo Árbol	2069 ± 33	2011 ± 19	2032 ± 3
	2006 ± 34	2022 ± 16	2033 ± 4
	1966 ± 35	2005 ± 30	2214 ± 255
	2038 ± 36	2039 ± 15	2022 ± 26
	1967 ± 37	2010 ± 22	2022 ± 16
Promedio	**2009 ± 35**	**2017 ± 20**	**2065 ± 61**

Montículos
Operación de Inserción



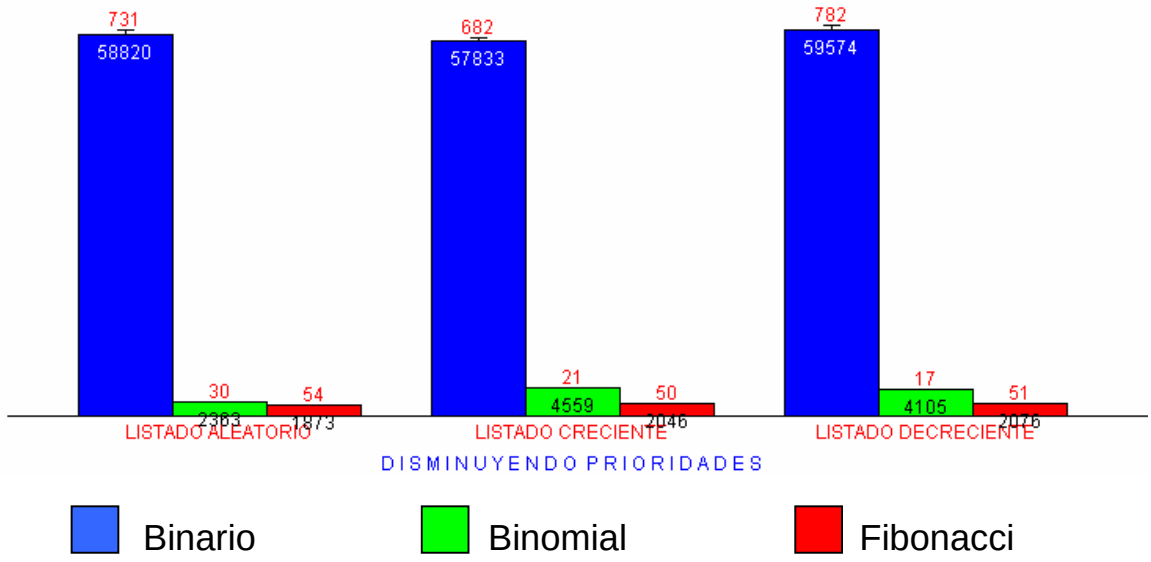
Montículos	**Operación afegeix**		
Núm. Elementos:	10000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Algoritmo Binario	2046 ± 75	2154 ± 59	2197 ± 124
	2051 ± 46	2070 ± 82	2055 ± 78
	2041 ± 79	2142 ± 47	2065 ± 66
	2120 ± 80	2000 ± 48	2214 ± 80
	2013 ± 59	2053 ± 31	2098 ± 64
Promedio	**2054 ± 68**	**2084 ± 53**	**2126 ± 82**
Algoritmo Binomial	1963 ± 148	1906 ± 21	1914 ± 16
	1813 ± 4	1881 ± 46	1874 ± 27
	1929 ± 161	1903 ± 29	1884 ± 21
	1891 ± 31	1934 ± 20	2014 ± 178
	1903 ± 23	1947 ± 31	1961 ± 22
Promedio	**1900 ± 73**	**1914 ± 29**	**1929 ± 53**
Algoritmo Fibonacci	1958 ± 33	1970 ± 26	1996 ± 37
	1868 ± 34	1917 ± 14	1804 ± 16
	1927 ± 35	1837 ± 30	1969 ± 26
	1984 ± 36	2008 ± 20	2011 ± 23
	1983 ± 37	2043 ± 52	2039 ± 24
Promedio	**1944 ± 35**	**1955 ± 28**	**1964 ± 25**

Montículos
Operación de extracción mínimos



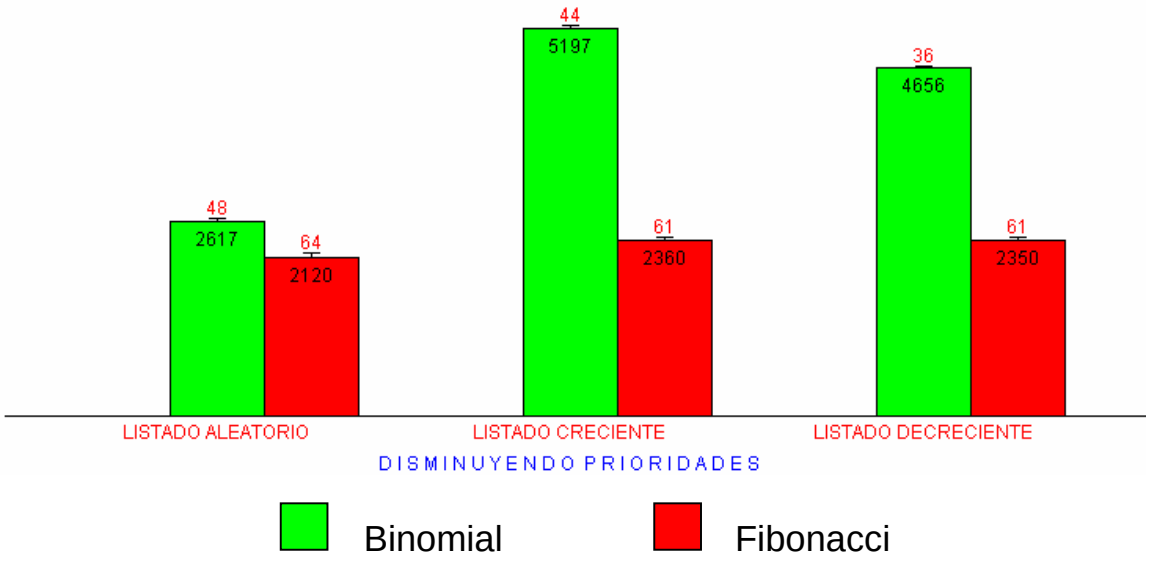
Montículos	**Operación extraeMinimo**		
Núm. Elementos:	10000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Algoritmo Binario	1813 ± 21	2049 ± 19	1433 ± 22
	1774 ± 23	2038 ± 20	1434 ± 22
	1825 ± 24	2088 ± 50	1436 ± 39
	1788 ± 29	2129 ± 26	1458 ± 48
	1857 ± 35	2046 ± 18	1436 ± 33
Promedio	**1811 ± 26**	**2070 ± 27**	**1439 ± 33**
Algoritmo Binomial	4181 ± 12	2868 ± 16	3602 ± 32
	4224 ± 13	2856 ± 28	3593 ± 21
	4226 ± 31	2847 ± 14	3581 ± 17
	4188 ± 25	2828 ± 21	3626 ± 71
	4205 ± 94	2807 ± 15	3184 ± 4
Promedio	**4205 ± 35**	**2841 ± 19**	**3517 ± 29**
Algoritmo Fibonacci	0 ± 33	17 ± 20	110 ± 0
	0 ± 34	22 ± 20	110 ± 0
	0 ± 35	0 ± 0	0 ± 0
	0 ± 36	12 ± 18	16 ± 19
	0 ± 37	54 ± 4	29 ± 22
Promedio	**0 ± 35**	**21 ± 12**	**53 ± 8**

Montículos
Operación de disminuir



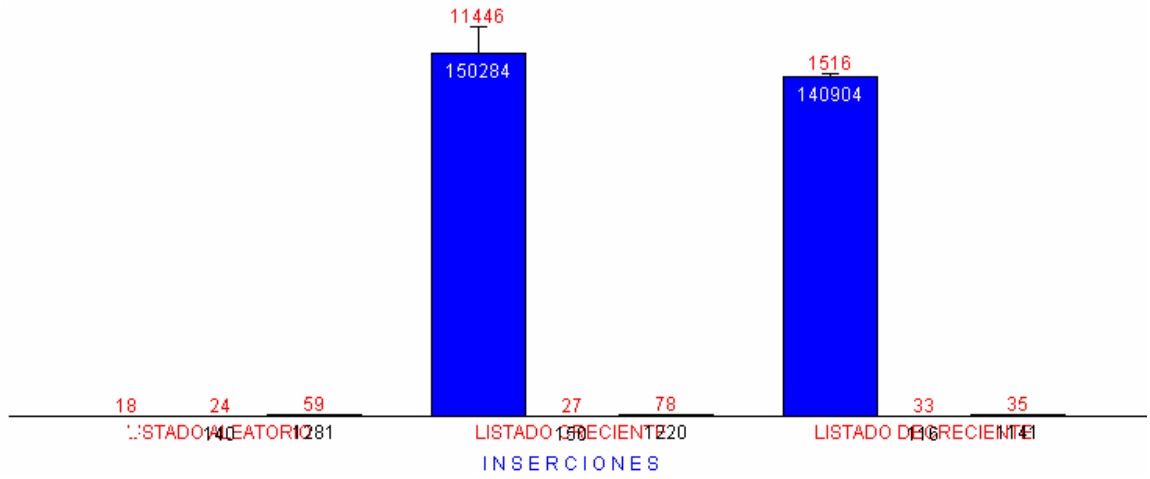
Montículos	**Operación extraeMinimo**		
Núm. Elementos:	10000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Algoritmo Binario	58820 ± 731	57833 ± 682	59574 ± 782
	0 ± 0	0 ± 0	0 ± 0
	0 ± 0	0 ± 0	0 ± 0
	0 ± 0	0 ± 0	0 ± 0
	0 ± 0	0 ± 0	0 ± 0
Promedio	**11764 ± 146**	**11567 ± 136**	**11915 ± 156**
Algoritmo Binomial	2363 ± 30	4559 ± 21	4105 ± 17
	0 ± 0	0 ± 0	0 ± 0
	0 ± 0	0 ± 0	0 ± 0
	0 ± 0	0 ± 0	0 ± 0
	0 ± 0	0 ± 0	0 ± 0
Promedio	**473 ± 6**	**912 ± 4**	**821 ± 3**
Algoritmo Fibonacci	1873 ± 33	2046 ± 50	2076 ± 51
	0 ± 34	0 ± 0	0 ± 0
	0 ± 35	0 ± 0	0 ± 0
	0 ± 36	0 ± 0	0 ± 0
	0 ± 37	0 ± 0	0 ± 0
Promedio	**375 ± 35**	**409 ± 10**	**415 ± 10**

Montículos
Operación de disminuir



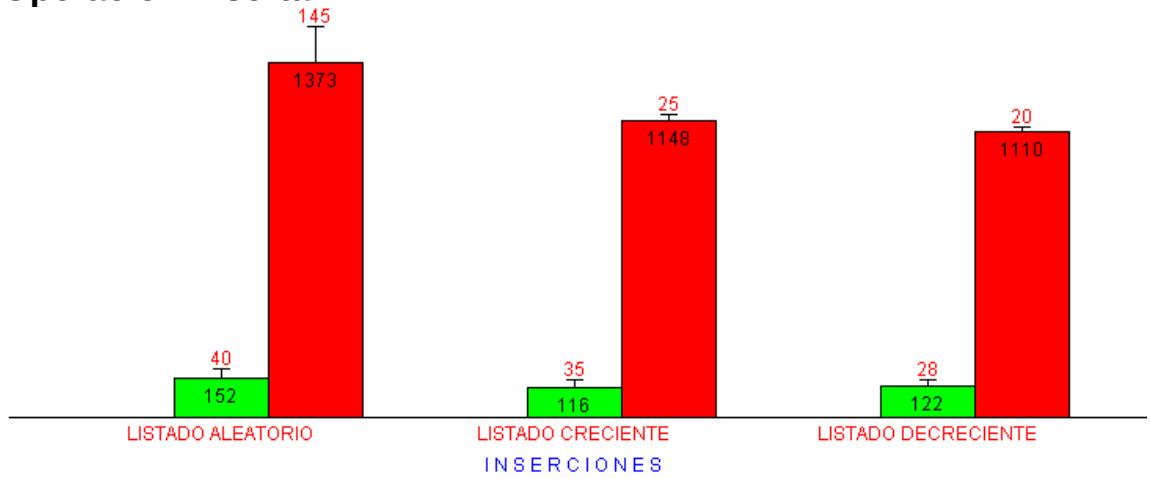
Montículos	**Operación extraeMinimo**		
Núm. Elementos:	10000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Algoritmo Binomial	2617 ± 48	5197 ± 44	4656 ± 36
	2867 ± 54	5327 ± 53	4691 ± 47
	2823 ± 55	5268 ± 34	5069 ± 294
	2838 ± 36	5572 ± 23	4687 ± 33
	2813 ± 43	5337 ± 32	4677 ± 26
Promedio	**2792 ± 47**	**5340 ± 37**	**4756 ± 87**
Algoritmo Fibonacci	2120 ± 33	2360 ± 61	2350 ± 61
	2156 ± 34	2396 ± 82	2367 ± 60
	2169 ± 35	2460 ± 86	2400 ± 51
	2241 ± 36	2454 ± 74	2390 ± 60
	2162 ± 37	2373 ± 48	2380 ± 85
Promedio	**2170 ± 35**	**2409 ± 70**	**2377 ± 63**

Árboles Binarios Búsqueda
Operación Insertar



Montículos	Operación Insertar		
Núm. Elementos:	10000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Árbol Encadenado	393 ± 18	150284 ± 11446	140904 ± 1516
	±	±	± 38
	±	±	± 29
	±	±	± 21
	±	±	± 14
Promedio	**393 ± 18**	**150284 ± 11446**	**140904 ± 324**
Árbol Red-Black	140 ± 24	150 ± 27	116 ± 33
	± 14	± 23	± 22
	± 4	± 24	± 22
	± 14	± 0	± 23
	± 19	± 24	± 22
Promedio	**140 ± 15**	**150 ± 20**	**116 ± 24**
Árbol AVL	1281 ± 33	1220 ± 78	1141 ± 35
	±	±	± 4
	±	±	± 4
	±	±	± 19
	±	±	± 3
Promedio	**1281 ± 33**	**1220 ± 78**	**1141 ± 13**

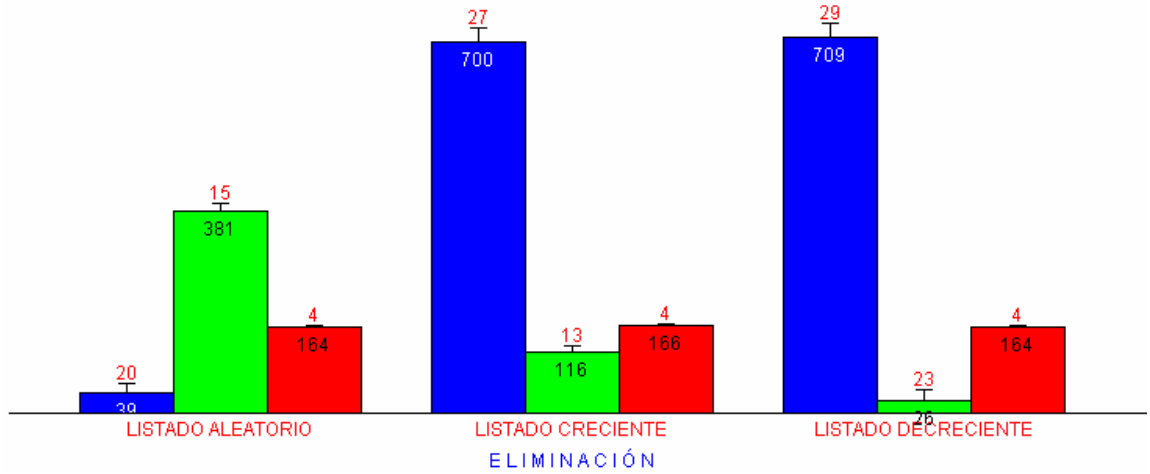
Árboles Binarios Búsqueda
Operación Insertar



Montículos	Operación Insertar		
Núm. Elementos:	10000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Árbol Red-Black	152 ± 40	116 ± 35	122 ± 28
	153 ± 29	150 ± 20	117 ± 15
	141 ± 45	129 ± 30	126 ± 33
	147 ± 30	110 ± 0	104 ± 24
	153 ± 25	123 ± 20	116 ± 25
Promedio	**149 ± 34**	**126 ± 21**	**117 ± 25**

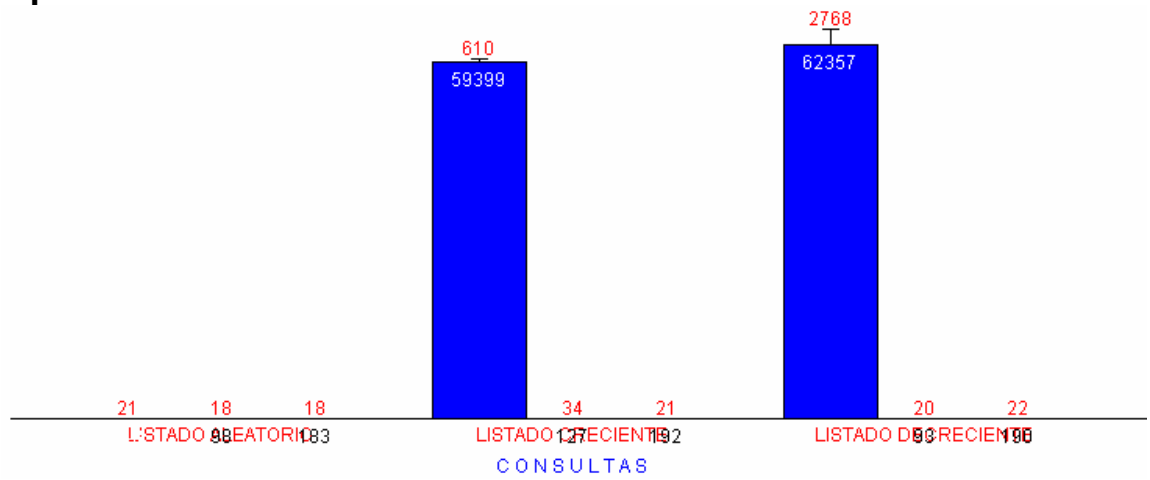
Árbol AVL	1373 ± 33	1148 ± 25	1110 ± 20
	1329 ± 34	1334 ± 96	1281 ± 88
	1227 ± 35	1136 ± 23	1122 ± 20
	1226 ± 36	1296 ± 203	1153 ± 56
	1240 ± 37	1118 ± 21	1121 ± 21
Promedio	**1279 ± 35**	**1206 ± 74**	**1157 ± 41**

Árboles Binarios Búsqueda
Operación Eliminar



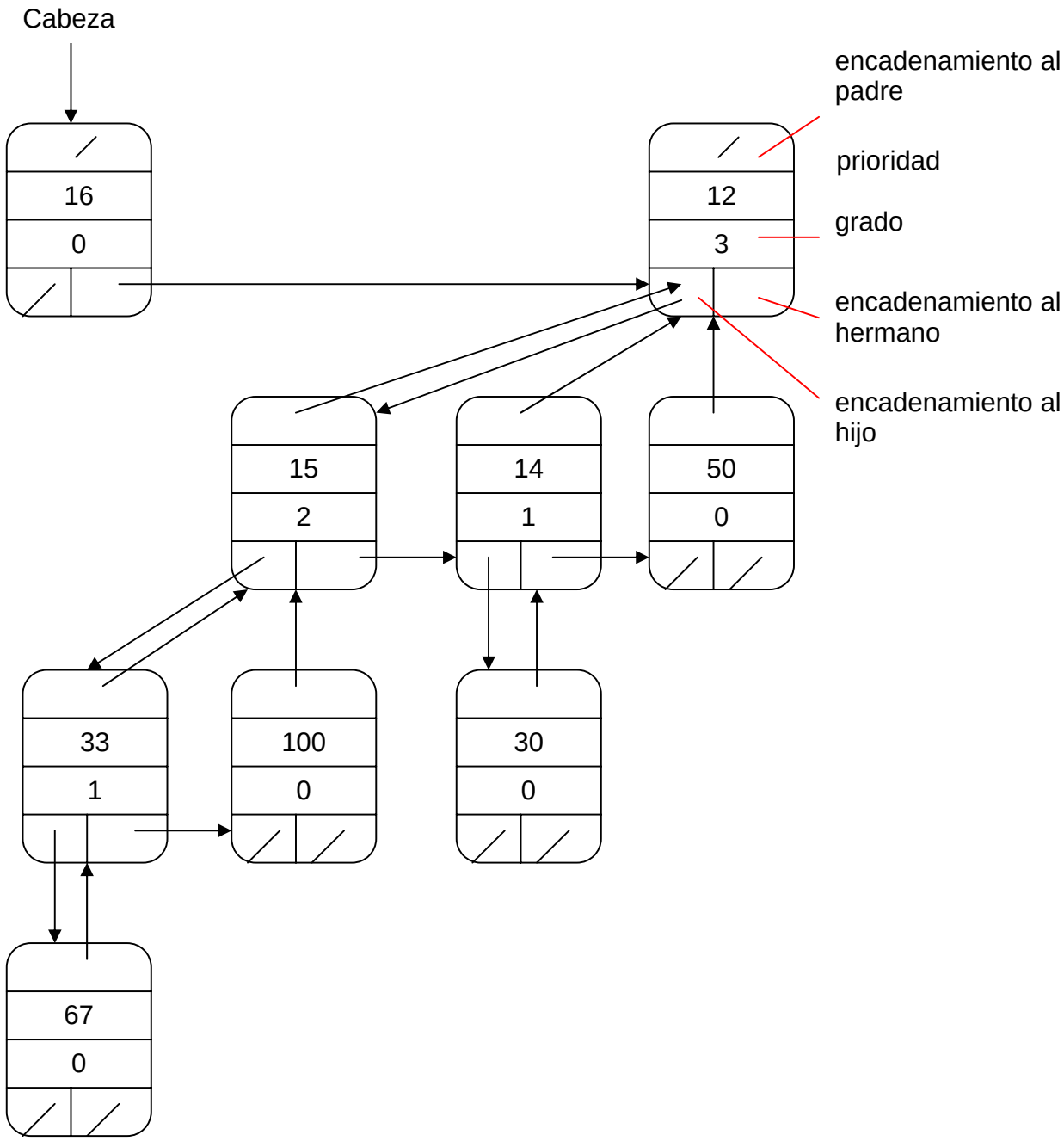
Montículos	Operación Eliminar		
Núm. Elementos:	2000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Árbol Encadenado	39 ± 20	700 ± 27	709 ± 29
	37 ± 18	927 ± 84	733 ± 38
	29 ± 22	717 ± 4	703 ± 29
	44 ± 17	721 ± 34	696 ± 21
	33 ± 21	701 ± 20	708 ± 14
Promedio	**36 ± 20**	**753 ± 34**	**710 ± 26**
Árbol Red-Black	381 ± 15	116 ± 13	26 ± 23
	378 ± 14	141 ± 23	24 ± 22
	386 ± 4	143 ± 24	24 ± 22
	378 ± 14	110 ± 0	31 ± 23
	397 ± 19	137 ± 24	30 ± 22
Promedio	**384 ± 13**	**129 ± 17**	**27 ± 22**
Árbol AVL	164 ± 33	166 ± 4	164 ± 4
	176 ± 34	165 ± 4	164 ± 4
	176 ± 35	165 ± 4	164 ± 4
	167 ± 36	165 ± 19	182 ± 19
	191 ± 37	164 ± 4	163 ± 3
Promedio	**175 ± 35**	**165 ± 7**	**167 ± 7**

Árboles Binarios Búsqueda
Operación Consultas

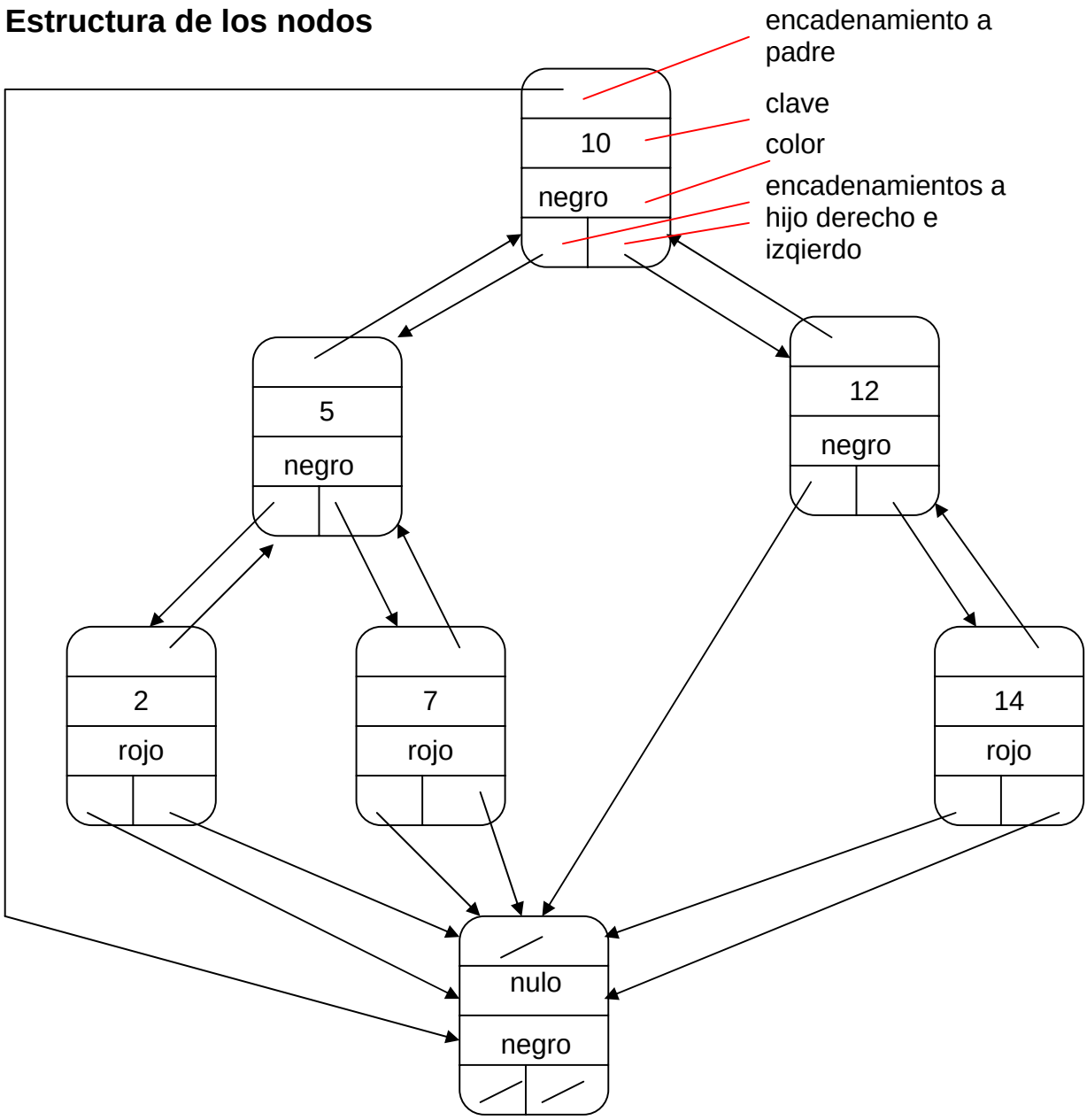


Montículos	Operación Consultar		
Núm. Elementos:	10000		
Núm. Repeticiones:	10		
	Elementos Orden aleatorios	**Elementos Orden Creciente**	**Elementos Orden Decreciente**
Árbol Encadenado	248 ± 21	59399 ± 610	62357 ± 2768
	±	±	± 38
	±	±	± 29
	±	±	± 21
	±	±	± 14
Promedio	**248 ± 21**	**59399 ± 610**	**62357 ± 574**
Árbol Red-Black	98 ± 18	127 ± 34	93 ± 20
	± 14	± 23	± 22
	± 4	± 24	± 22
	± 14	± 0	± 23
	± 19	± 24	± 22
Promedio	**98 ± 14**	**127 ± 21**	**93 ± 22**
Árbol AVL	183 ± 33	192 ± 21	190 ± 22
	±	±	± 4
	±	±	± 4
	±	±	± 19
	±	±	± 3
Promedio	**183 ± 33**	**192 ± 21**	**190 ± 10**

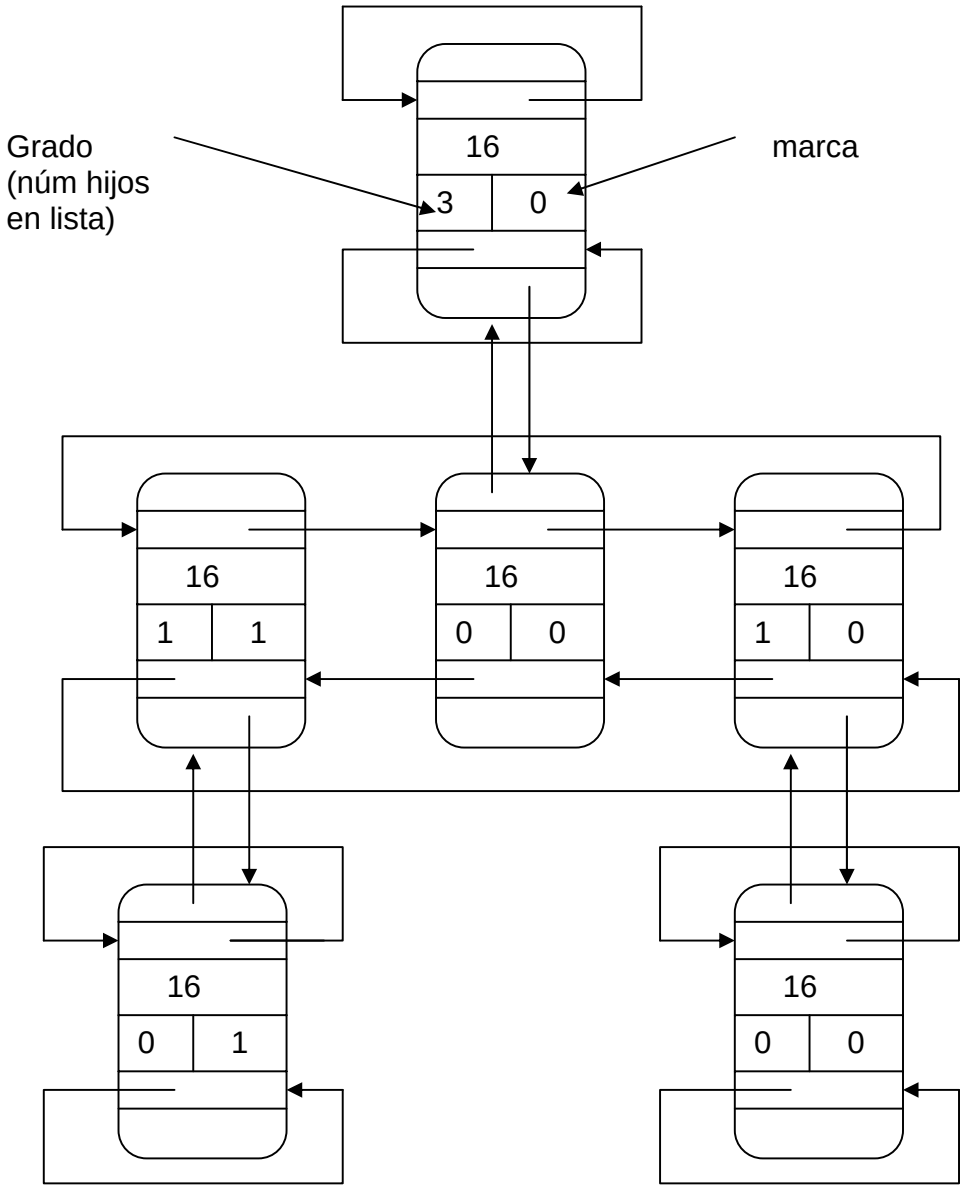
Montículo Binomial
Estructura de los nodos



Árbol Red-Black
Estructura de los nodos



Montículo Fibonacci
Estructura de los nodos



Ejemplo lectura datos de Archivo Evaluación, leído con Excel y su interpretación.

Arboles Benchmarcks **<--- Tipo de estructura evaluada**
E L I M I N A C I Ó N **<--- Operación evaluada**
2000 **<--- Número de elementos**
10 **<--- Número repeticiones**
28 **<--- Estructura 1 - Lista Aleatoria - Media**
21 **<--- Estructura 1 - Lista Aleatoria - Intervalo**
337 **<--- Estructura 2 - Lista Aleatoria - Media**
15 **<--- Estructura 2 - Lista Aleatoria - Intervalo**
166 **<--- Estructura 3 - Lista Aleatoria - Media**
4 **<--- Estructura 3 - Lista Aleatoria - Intervalo**
658 **<--- Estructura 1 - Lista Creciente - Media**
3 **<--- Estructura 1 - Lista Creciente - Intervalo**
143 **<--- Estructura 2 - Lista Creciente - Media**
19 **<--- Estructura 2 - Lista Creciente - Intervalo**
166 **<--- Estructura 3 - Lista Creciente - Media**
4 **<--- Estructura 3 - Lista Creciente - Intervalo**
667 **<--- Estructura 1 - Lista Decreciente - Media**
15 **<--- Estructura 1 - Lista Decreciente - Intervalo**
151 **<--- Estructura 2 - Lista Decreciente - Media**
18 **<--- Estructura 2 - Lista Decreciente - Intervalo**
164 **<--- Estructura 3 - Lista Decreciente - Media**
4 **<--- Estructura 3 - Lista Decreciente - Intervalo**
---- detalle ----
10 **<--- Estructura 1 - Lista Aleatoria - Valores**
60
0
0
50
50
0
60
60
0
0
Validos: 10/10 **<--- Resumen resultados**
Media datos: 28.0
Desviacion Estandar datos: 29.73961069759395
Media final: 28.0
Desviacion Estandar final: 29.73961069759395
Intervalo: 6.727042142663848-49.27295785733615
---
660 **<--- Estructura 2 - Lista Aleatoria - Valores**
330
330
330
330

330
330
330
390
330
Validos: 9/10 **<--- Resumen resultados**
Media datos: 369.0
Desviacion Estandar datos: 103.97114984456024
Media final: 336.6666666666667
Desviacion Estandar final: 20.0
Intervalo: 321.29333333333335-352.04
---
170 **<--- Estructura 3 - Lista Aleatoria - Valores**
160
170
170
160
170
170
160
160
170
Validos: 10/10 **<--- Resumen resultados**
Media datos: 166.0
Desviacion Estandar datos: 5.163977794943222
Media final: 166.0
Desviacion Estandar final: 5.163977794943222
Intervalo: 162.30616946788297-169.69383053211703

1260 **<--- Estructura 1 - Lista Creciente - Valores**
660
660
650
660
660
660
660
650
660
Validos: 9/10 **<--- Resumen resultados**
Media datos: 718.0
Desviacion Estandar datos: 190.48476171191345
Media final: 657.7777777777778
Desviacion Estandar final: 4.409585518440984
Intervalo: 654.3882763759362-661.1672791796194
---
830 **<--- Estructura 2 - Lista Creciente - Valores**
110
160

160
160
110
110
160
160
160
Validos: 9/10 **<--- Resumen resultados**
Media datos: 212.0
Desviacion Estandar datos: 218.41855843006257
Media final: 143.33333333333334
Desviacion Estandar final: 25.0
Intervalo: 124.11666666666667-162.55
---
170 **<--- Estructura 3 - Lista Creciente - Valores**
160
170
220
160
170
160
160
170
170
Validos: 9/10 **<--- Resumen resultados**
Media datos: 171.0
Desviacion Estandar datos: 17.919573407620817
Media final: 165.55555555555554
Desviacion Estandar final: 5.270462766947299
Intervalo: 161.50432650869539-169.6067846024157

1270 **<--- Estructura 1 - Lista Dereciente - Valores**
660
660
660
660
660
660
660
660
720
Validos: 9/10 **<--- Resumen resultados**
Media datos: 727.0
Desviacion Estandar datos: 191.720282356006
Media final: 666.6666666666666
Desviacion Estandar final: 20.0
Intervalo: 651.2933333333333-682.04
---
710 **<--- Estructura 2 - Lista Dereciente - Valores**

160
170
170
110
160
160
160
110
160
Validos: 9/10 **<--- Resumen resultados**
Media datos: 207.0
Desviacion Estandar datos: 178.1416415228187
Media final: 151.11111111111111
Desviacion Estandar final: 23.687784005919827
Intervalo: 132.90310113856074-169.31912108366149
---
170 **<--- Estructura 3 - Lista Dereciente - Valores**
160
160
170
160
160
170
160
160
170
Validos: 10/10 **<--- Resumen resultados**
Media datos: 164.0
Desviacion Estandar datos: 5.163977794943222
Media final: 164.0
Desviacion Estandar final: 5.163977794943222
Intervalo: 160.30616946788297-167.69383053211703
---

8. Bibliografía

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein. *Introduction to Algorithms, Second Edition.* The MIT Press, 2001.

[2] Xavier Franch Gutiérrez. *Estructura de datos. Especificación, diseño e implementaciones.* Ediciones UPC.

[3] Xavier Franch Gutiérrez. *Estructura de la informació. Tercera edición.* Universitat Oberta de Catalunya, 2001.

[4] Y. Lacroix. *Analyse chimique. Interprétation des resultants par le calcul statistique.* Masson et Cia, Editeurs. Paris, 1962

[5] Agustín Froute. *Java 2. Manual de usuario y tutorial, Segunda edición.* RA-MA Editorial, Madrid, 2000.

[6] Alfred V. Aho, John E. Hopcroft y Jeffrey D. Ullman. *Estructura de datos y algoritmos.* Addison-Wesley Iberoamericana, México.