

IDENTIFICACIÓN DEL PROBLEMA

Identificación de necesidades y síntomas

- El software debe permitir el manejo de información de gran tamaño y permitir el ingreso de datos.
- La información a manejar son los datos más relevantes de los jugadores profesionales de baloncesto en el planeta.
- Los datos almacenados deben tener la posibilidad de ser eliminados, modificados y consultados con base a criterios de búsqueda.
- Se necesita que el software sea altamente eficiente en la rapidez para obtener datos almacenados, por lo tanto la complejidad temporal en consultas no puede ser lineal.
- La información debe ser guardada en memoria secundaria debido a su magnitud.

Definición del problema

La federación internacional de baloncesto, FIBA, requiere una aplicación que le permita almacenar los datos relevantes de todos los jugadores de baloncesto en la categoría profesional y su vez que se puedan realizar operaciones como consultas, edición, inserción y eliminación sobre dichos datos.

RECOPIACIÓN DE INFORMACIÓN

Definiciones

-Coyuntura:Conjunto de circunstancias que intervienen en la resolución de un asunto importante.

(Fuente: <http://www.wordreference.com/definicion/coyuntura>)

-Archivos Cvs: Los archivos CSV (del inglés *comma-separated values*) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas (o punto y coma en donde la coma es el separador decimal: Chile, Perú, Argentina, España, Brasil...) y las filas por saltos de línea.

El formato CSV es muy sencillo y no indica un juego de caracteres concreto, ni cómo van situados los bytes, ni el formato para el salto de línea. Estos puntos deben indicarse muchas veces al abrir el archivo, por ejemplo, con una hoja de cálculo.

(Fuente: https://es.wikipedia.org/wiki/Valores_separados_por_comas)

-Base de datos: Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. Actualmente, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos están en formato digital, siendo este un componente electrónico, por tanto se ha desarrollado y se ofrece un amplio rango de soluciones al problema del almacenamiento de datos.

(Fuente: https://es.wikipedia.org/wiki/Base_de_datos)

-Árbol binario: Usualmente el árbol se trata como conjunto dinámico, mediante la creación de sus nodos bajo demanda. Es decir, un nodo se crea con malloc, y contiene punteros a otros nodos de la estructura. En caso de un árbol binario, debe disponerse al menos de dos punteros. Se ilustra un ejemplo con una clave entera, no se muestra espacio para la información periférica que puede estar

asociada al nodo. En la implementación de algunas operaciones conviene disponer de un puntero al padre del nodo, que tampoco se declara en el molde del nodo.

(Fuente: <http://www2.elo.utfsm.cl/~lsb/elo320/clases/c6.pdf>)

SOLUCIONES CREATIVAS

Se requiere el uso de árboles binarios para el desarrollo de la solución.

- Alternativa 1: árbol binario de búsqueda.

Para cada nodo de un árbol binario de búsqueda debe cumplirse la propiedad: Las claves de los nodos del subárbol izquierdo deben ser menores que la clave de la raíz. Las claves de los nodos del subárbol derecho deben ser mayores que la clave de la raíz. a l D a l D Figura 6.3. Árbol binario de búsqueda. Esta definición no acepta elementos con claves duplicadas. el descendiente del subárbol izquierdo con mayor clave y el descendiente del subárbol derecho con menor valor de clave; los cuales son el antecesor y sucesor de la raíz. El siguiente árbol no es binario de búsqueda, ya que el nodo con clave 2, ubicado en el subárbol derecho de la raíz, tiene clave menor que ésta.

(Fuente: <http://www2.elo.utfsm.cl/~lsb/elo320/clases/c6.pdf>)

- Alternativa 2: árbol balanceado AVL.

Un árbol AVL es un árbol binario de búsqueda que cumple con la condición de que la diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho 1. La denominación de árbol AVL viene dada por los creadores de tal estructura (Adelson-Velskii y Landis). Recordamos que un árbol binario de búsqueda es un árbol binario en el cual cada nodo cumple con que todos los nodos de su subárbol izquierdo son menores que la raíz y todos los nodos del subárbol derecho son mayores que la raíz. Recordamos también que el tiempo de las operaciones sobre un árbol binario de búsqueda son $O(\log n)$ promedio, pero el peor caso es $O(n)$, donde n es el número de elementos. La propiedad de equilibrio que debe cumplir un árbol para ser AVL asegura que la profundidad del árbol sea $O(\log(n))$, por lo que las operaciones sobre estas estructuras no deberán recorrer mucho para hallar el elemento deseado. Como se verá, el tiempo de ejecución de las operaciones sobre estos árboles es, a lo sumo $O(\log(n))$ en el peor caso, donde n es la cantidad de elementos del árbol.

(Fuente: <http://es.tldp.org/Tutoriales/doc-programacion-arboles-avl/avl-trees.pdf>)

- Alternativa 3: árbol balanceado rojinegro.

Los árboles rojinegros son estructuras basadas en árboles binarios balanceados. F Un árbol rojinegro es un árbol de búsqueda binaria que satisface las siguientes propiedades: " Cada nodo o es rojo o es negro Cada hoja (nil) es negra Æ Si un nodo es rojo, entonces, sus hijos son negros Ø Cada camino de un nodo a cualquier descendiente tiene la misma cantidad de nodos negros.

(Fuente: <http://delta.cs.cinvestav.mx/~adiaz/anadis/RedBlackTree.pdf>)

TRANSICIÓN DE LAS IDEAS A LOS DISEÑOS PRELIMINARES

La revisión cuidadosa de las alternativas conduce a:

Alternativa 1: árbol de búsqueda binaria.

- Permite hacer búsqueda de rango de manera eficiente.
- Facilita la operación de persistencia de la información.

- Operaciones básicas como insertar, borrar y buscar, toman un tiempo proporcional a la altura del árbol.
- Es más eficiente en árboles de altura pequeña.
- Las operaciones básicas toman un tiempo de $O(\log n)$, pero para el peor caso es $O(n)$.
- El número de accesos al árbol es menor que en una lista enlazada.

Alternativa 2: árbol balanceado AVL.

- El árbol está algo equilibrado, de modo que es más eficiente que el árbol binario de búsqueda.
- La propiedad de equilibrio que debe cumplir un árbol para ser AVL asegura que la profundidad del árbol sea $O(\log(n))$.
- La complejidad de una búsqueda se mantiene siempre en orden de complejidad $O(\log n)$.
- Permite las operaciones básicas como insertar y borrar.
- La propiedad de equilibrio de los árboles AVL implica una dificultad a la hora de insertar o eliminar elementos.

Alternativa 3: árbol balanceado rojinegro.

- Las condiciones sobre los colores de los nodos garantizan que la profundidad de ninguna hoja es más del doble que la de ninguna otra, de este modo el árbol permanece balanceado.
- Las operaciones de búsqueda pueden implementarse en tiempo $O(\log n)$ para árboles rojinegros con n nodos.
- Las operaciones de inserción y eliminación pueden realizarse en tiempo $O(\log n)$ sin que el árbol deje de ser rojinegro, pero para ello se deben hacer otras operaciones adicionales, las cuales le dan un grado más de dificultad.

EVALUACIÓN Y SELECCIÓN DE LA MEJOR SOLUCIÓN

A continuación se evalúan cada una de las alternativas, teniendo en cuenta que la de mayor puntaje es la más eficiente.

Criterios:

Criterio A: La alternativa es la adecuada para solucionar alguno de los problemas que plantea la situación.

- [2] Ayuda en la solución
- [1] No ayuda en la solución

Criterio B: La alternativa es eficiente a un nivel:

- [4] Constante.
- [3] Mayor a constante.
- [2] Logarítmica
- [1] Lineal o mayor

Criterio C: La alternativa permite combinarse con otras para generar una mejor solución

- [1] Mejora con otra alternativa
- [2] Es suficiente por ella misma.

Criterio D: La alternativa utiliza recursión.

- [1] En todos su métodos.
- [2] No usa recursión
- [3] Usa recursión, pero admite iteración.

Evaluación de alternativas:

	Criterio A	Criterio B	Criterio C	Criterio D	Total
Alternativa 1	2	2	2	3	9
Alternativa 2	2	2	2	3	9
Alternativa 2	2	2	2	3	9

Selección:

De acuerdo con la evaluación anterior, todas las alternativas consideradas se ajustan al problema, todas las alternativas se pueden usar para resolver una parte del problema.

PREPARACIÓN DE LOS INFORMES Y ESPECIFICACIONES

Especificación del problema:

Problema: Leer información de gran tamaño o ingresada por usuario.

Entrada: archivos csv o datos ingresados por usuario

A_n , donde $A_n \in B$ (conjunto de datos csv)

Salida: archivos de textos con la información ingresada.

A_n , donde A_n es un elemento de B (conjunto de datos csv) o un elemento con información ingresada..

Pseudocódigo para consultas en un árbol binario.

TREE-SEARCH(x,k)

if x=nil or k=key[x]

then return x

if k<key[x]

then return TREE-SEARCH(left[x],k)

```
else return TREE-SEARCH(right[x],k)
```

Pseudocódigo para inserción en un árbol binario.

```
TREE-INSERT(x)
```

```
Y ← nil
```

```
X ← root[T]
```

```
while x ≠ nil
```

```
do y ← x
```

```
    if key[z] < key[x]
```

```
    then x ← left[x]
```

```
    else x ← right[x]
```

```
p[z] ← y
```

```
if y = nil
```

```
then root[T] ← z
```

```
else if key[z] < key[y]
```

```
    then left[y] ← z
```

```
    else right[y] ← z
```

Pseudocódigo para eliminación en un árbol binario.

```
TREE-DELETE(x)
```

```
if left[z] = nil or right[z] = nil
```

```
then y ← z
```

```
    else y ← TREE-SUCCESSOR(z)
```

```
if left[y] ≠ nil
```

```
then x ← left[y]
```

```
else x ← right[y]
```

```
if x ≠ nil
```

```
then p[x] ← p[y]
```

```
if p[y] = nil
```

```
then root[T] ← x
```

```
else if y = left[p[y]]
```

```
    then left[p[y]] ← x
```

```
    else right[p[y]] ← x
```

```
if y ≠ z
```

```
then key[z] ← key[y]
```

```
return y
```

Pseudocódigo para eliminación en árbol rojinegro

```
RB-TREE-DELETE(T, z)
```

```
if left[z] = nil [T]
```

```
or right[z] = nil [T]
```

```
    y = z
```

```

else
    y = TREE-SUCCESSOR(z)
if left[y] != nil[T]
    x = left[y]
else
    x = right[y]
p[x] = p[y]
if p[y] = nil[t]
    root[T] = x
else
    if y = left[p[y]]
        left[p[y]] = x
    else
        right[p[y]] = x
if y != z
    key[z] = key[y]
if color[y] = BLACK
    RB-DELETE-FIXUP(T, x)
return y

```

Pseudocódigo para Búsqueda en árbol rojinegro

```

RB-TREE-SEARCH(x, k)
if (x = nil) or (k = key[x])
    return x
if (k < key[x])
    return RB-TREE-SEARCH(left[x], k)
else
    return RB-TREE-SEARCH(right[x], k)

```

Pseudocódigo para inserción en árbol rojinegro

```

TREE-INSERT(T, z)
y = nil
x = root[T]
while x != nil
    y = x
    if key[z] < key[x]
        x = left[x]
    else
        x = right[x]
p[z] = y

```

```

if y = nil
    root [ T ] = z
else
    if key [ z ] < key [ y ]
        left [ y ] = z
    else
        right [ y ] = z
left [ z ] = nil [ T ]
right [ z ] = nil [ T ]
color [ z ] = RED
RB-INSERT-FIX-UP(T, z)

```

Pseudocódigo para método auxiliar de inserción en árbol rojinegro

```

RB-INSERT-FIX-UP(T, z)
    while color [ p [ z ] ] = RED
        if p [ z ] = left [ p [ p [ z ] ] ]
            y = right [ p [ p [ z ] ] ]
            if color [ y ] = RED
                color [ p [ z ] ] = BLACK
                color [ y ] = BLACK
                color [ p [ p [ z ] ] ] = RED
                z = p [ p [ x ] ]
            else
                if z = right [ p [ z ] ]
                    z = p [ z ] // Caso 2
                RB-LEFT-ROTATE(T, z)
                color [ p [ z ] ] = BLACK
                color [ p [ p [ z ] ] ] = RED
                RB-RIGHT-ROTATE(T, p [ p [ z ] ] )
        else
            procedimiento simétrico cambiando "right" por "left"
    color [ root [ T ] ] = BLACK

```

Pseudocódigo para método insertar en árbol AVL

- 1 AVL-Insert(T, k)
- 2 begin
- 3 Insertar k en T como en un ABB
- 4 Sea x el nodo agregado

```

5     AVL-Rebalance(T, x)
6 end

```

Pseudocódigo para método eliminar en árbol AVL

```

1 AVL-Delete(T, k)
2 begin
3     Borrar k de T como en un ABB
4     Sea y el primer nodo posiblemente desbalanceado
5     AVL-Rebalance(T, y)
6 end

```

Pseudocódigo para método auxiliar de inserción y eliminación en árbol AVL

```

1 AVL-Rebalance(T, x)
2 begin
3     foreach p nodo en el camino entre x y la raíz de T do
4         if |fb(p)| > 1 then
5             Rebalancear el subarbol con raíz en p como en alguno de los
casos A,..., F
6         end
7     end
8 end

```

Implementación del diseño

Implementación en lenguaje de programación

Lista de tareas a implementar:

- Buscar un jugador guardado en la base de datos.
- Eliminar un jugador:
- Agregar un nuevo jugador
- Permitir la persistencia de la información de los jugadores.
- Dar jugadores con valor de criterio igual a un jugador dado.
- Dar jugadores con valor de criterio menor a un jugador dado

Eliminar un jugador:

Nombre	delete
Descripción	Este método elimina un jugador de la base de datos
Entrada	Jugador a eliminar
Retorno	-----

```
public void delete(Player elem) {
```



```

if(elem.getType() == 0) {

    players.deleteRB(elem);

} else if(elem.getType() == 1) {

    try {
        gamesT.eliminar(elem);
    } catch (ElementoNoExisteException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

} else if(elem.getType() == 2) {

    try {
        mpT.eliminar(elem);
    } catch (ElementoNoExisteException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

} else if(elem.getType() == 3) {

    perT.deleteRB(elem);

} else if(elem.getType() == 4) {

    tsT.deleteRB(elem);

} else if(elem.getType() == 5) {

    ftrT.deleteRB(elem);

}

}

```

Agregar un nuevo jugador

Nombre	addnewPlayer
Descripción	Este método agrega un nuevo jugador a la base de datos Fiba
Entrada	Un jugador con datos específicos.
Retorno	-----

```
public void addNewPlayer(Player p) {  
  
    if(p.getType() == 0) {  
  
        players.insertRB(p);  
  
    } else if(p.getType() == 1) {  
  
        try {  
            gamesT.insertar(p);  
        } catch (ElementoExisteException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
  
    } else if(p.getType() == 2) {  
  
        try {  
            mpT.insertar(p);  
        } catch (ElementoExisteException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
  
    } else if(p.getType() == 3) {  
  
        perT.insertRB(p);  
  
    } else if(p.getType() == 4) {  
  
        tsT.insertRB(p);  
  
    } else if(p.getType() == 5) {  
  
        ftrT.insertRB(p);  
  
    }  
}
```

}

}

Permitir la persistencia de la información de los jugadores.

Nombre	addPlayerDefault
Descripción	Este método se encarga de que la información de los jugadores sea persistente al momento de iniciar el programa.
Entrada	Sin entradas
Retorno	Sin salidas

```
public void addPlayerDefault() {  
  
    BufferedReader br = null;  
  
    try {  
        br = new BufferedReader(new FileReader("src/main/java/data/data.csv"));  
        String line = br.readLine();  
        line = br.readLine();  
  
        while(line != null) {  
            String[] fields = line.split(",");  
            String year = fields[0];  
            String team = fields[1];  
            String name = fields[2];  
            int age = Integer.parseInt(fields[3]);  
            int games = Integer.parseInt(fields[4]);  
            int mp = Integer.parseInt(fields[5]);  
            double per = Double.parseDouble(fields[6]);  
  
            double ts = 0, ftr = 0;  
  
            if(!fields[7].isEmpty()) {  
                ts = Double.parseDouble(fields[7]);  
            }  
            if(!fields[9].isEmpty()) {  
                ftr = Double.parseDouble(fields[9]);  
            }  
  
            Player player = new Player(year, team, name, age, games, mp, per, ts, ftr,  
0);  
  
            players.insertRB(player);  
            player = new Player(year, team, name, age, games, mp, per, ts, ftr, 1);  
            gamesT.insertar(player);  
            player = new Player(year, team, name, age, games, mp, per, ts, ftr, 2);  
            mpT.insertar(player);  
            player = new Player(year, team, name, age, games, mp, per, ts, ftr, 3);  
            perT.insertRB(player);  
        }  
    }  
}
```

```

        player = new Player(year, team, name, age, games, mp, per, ts, ftr, 4);
        tsT.insertRB(player);
        player = new Player(year, team, name, age, games, mp, per, ts, ftr, 5);
        ftrT.insertRB(player);
        line = br.readLine();

    }

} catch (Exception e) {

    e.printStackTrace();
} finally {

    try {
        br.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

}

}

```

Dar jugadores con valor de criterio igual a un jugador dado

Nombre	getSame
Descripción	Este método retorna los jugadores con valor de criterio igual al de uno dado
Entrada	Jugador a comparar
Retorno	Lista con jugadores con valor de criterio menor a jugador en cuestión

```

public void getSame(Player player){

    if(player.getType() == 3) {

        try {
            perT.getRoot().getNode(player);
            perT.getSame(player);

        } catch (ElementoNoExisteException e) {

            perT.insertRB(player);

        }

    }

}

```

```

        perT.getSame(player);
        perT.deleteRB(player);

    }

} else if(player.getType() == 4) {

    try {
        tsT.getRoot().getNode(player);
        tsT.getSame(player);

    } catch (ElementoNoExisteException e) {

        tsT.insertRB(player);
        tsT.getSame(player);
        tsT.deleteRB(player);

    }

} else if(player.getType() == 5) {

    try {
        ftrT.getRoot().getNode(player);
        ftrT.getSame(player);

    } catch (ElementoNoExisteException e) {

        ftrT.insertRB(player);
        ftrT.getSame(player);
        ftrT.deleteRB(player);

    }

}

}

```

Buscar un jugador guardado

Este ítem está apoyado con dos métodos, por lo tanto se generaliza.

Nombre	Buscar un jugador
Descripción	Este método busca un jugador guardado
Entrada	Jugador a encontrar

Retorno	Jugador
----------------	---------

Buscar en árbol AVL

```
public T buscar( T modelo )
{
    return ( raiz != null ) ? raiz.buscar( modelo ) : null;
}
```

Buscar en árbol RojiNegro

```
public RedBlackNode<A> search(A elem, RedBlackNode<A> r) {
    if(root== null) {
        return null;
    }

    if(r != null) {

        if(r.getInfoNode().compareTo(elem) == 0) {
            return r;
        }else if(r.getInfoNode().compareTo(elem) < 0) {
            return search(elem, r.getRChild());

        }else {
            return search(elem, r.getLChild());
        }
    }else {
        return null;
    }

}
```

Dar jugadores con valor de criterio menor a un jugador dado

Nombre	getLess
Descripción	Este método retorna los jugadores con valor de criterio menor al de uno dado
Entrada	Jugador a comparar
Retorno	Lista con jugadores con valor de criterio menor a jugador en cuestión

```
public void getLess(Player player){

    if(player.getType() == 3) {
```

```

        try {
            perT.getRoot().getNode(player);
            perT.getLess(player);

        } catch (ElementoNoExisteException e) {

            perT.insertRB(player);
            perT.getLess(player);
            perT.deleteRB(player);

        }

    } else if(player.getType() == 4) {

        try {
            tsT.getRoot().getNode(player);
            tsT.getLess(player);

        } catch (ElementoNoExisteException e) {

            tsT.insertRB(player);
            tsT.getLess(player);
            tsT.deleteRB(player);

        }

    } else if(player.getType() == 5) {

        try {
            ftrT.getRoot().getNode(player);
            ftrT.getLess(player);

        } catch (ElementoNoExisteException e) {

            ftrT.insertRB(player);
            ftrT.getLess(player);
            ftrT.deleteRB(player);

        }

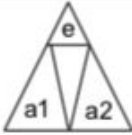
    }

}

```

TAD de las estructuras utilizadas

TAD Árbol binario

TAD ArbinOr[TipoAO]		
		
{ inv: a1 y a2 son disyuntos, todos los elementos de a1 son menores que e, todos los elementos de a2 son mayores que e, a1 y a2 son ordenados }		
Constructora: • inicArbinOr: → ArbinOr		
Modificadoras: • insArbinOr: ArbinOr x TipoAO → ArbinOr • elimArbinOr: ArbinOr x TipoAO → ArbinOr		
Analizadora: • estaArbinOr: ArbinOr x TipoAO → int		
ArbinOr inicArbinOr(void) /* Crea un árbol binario ordenado vacío */ { post: inicArbinOr = \emptyset }		
ArbinOr insArbinOr(ArbinOr a, TipoAO elem) /* Agrega un elemento a un árbol binario ordenado */ { pre: elem \notin a, a = A } { post: insArbinOr = $A \cup \{ elem \}$ }		
ArbinOr elimArbinOr(ArbinOr a, TipoAO elem) /* Elimina un elemento de un árbol binario ordenado */ { pre: elem \in a, a = A } { post: elimArbinOr = $A - \{ elem \}$ }		
int estaArbinOr(ArbinOr a, TipoAO elem) /* Informa si un elemento se encuentra en un árbol binario ordenado */ { post: estaArbinOr = (elem \in a) }		


```

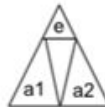
AVL elimAVL ( AVL a, TipoAVL elem )
/* Elimina un elemento de un árbol AVL */

{ pre: a = A, elem ∈ a }
{ post: elimAVL = A - { elem } }

```

TAD Árbo IAVL

TAD AVL[TipoAVL]



{ inv: a1 y a2 son disyuntos, todos los elementos de a1 son menores que e,
 todos los elementos de a2 son mayores que e, a1 y a2 son ordenados,
 $| altura(a1) - altura(a2) | \leq 1$, a1 y a2 son AVL }

Constructora:

- inicAVL: → AVL

Modificadoras:

- insAVL: AVL x TipoAVL → AVL
- elimAVL: AVL x TipoAVL → AVL

```

AVL inicAVL ( void )
/* Crea un árbol AVL vacío */

```

{ post: inicAVL = \emptyset }

```

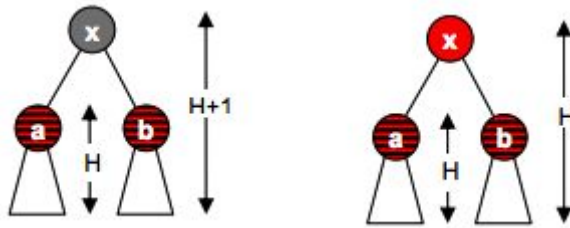
AVL insAVL ( AVL a, TipoAVL elem )
/* Adiciona un elemento a un árbol AVL */

```

{ pre: a = A, elem ∉ a }
 { post: insAVL = $A \cup \{ elem \}$ }

TAD Árbol Rojo-Negro

TAD Árbol Rojo-Negro



$$H(n) = \begin{cases} \text{máx}(H(n.\text{izdo}), H(n.\text{dcho})) + 1 & \text{si } n \text{ es negro} \\ \text{máx}(H(n.\text{izdo}), H(n.\text{dcho})) & \text{si } n \text{ es rojo} \end{cases}$$

- ArbolRojoNegro - ArbolRojoNegro
 - insert Element ArbolRojoNegro
 - RBInsertFixUp NodoArbolRojoNegro ArbolRojoNegro
 - RBDeleteFixUp NodoArbolRojoNegro ArbolRojoNegro
 - deleteItem Element ArbolRojoNegro
 - search Element NodoArbolRojoNegro
 - leftRotate Element-Value NodoArbolRojoNegro
 - rightRotate NodoArbolRojoNegro Element